



***System Architecture
Reference Guide***

Rev. 21.0

DOC9473-2LA

System Architecture Reference Guide

Second Edition

by

Marilyn Hammond

Prime Computer, Inc.
Prime Park
Natick, Massachusetts 01760

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer, Inc. Prime Computer, Inc., assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1987 by Prime Computer, Inc. All rights reserved.

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc. DISCOVER, INFO/BASIC, INFORM, MIDAS, MIDASPLUS, PERFORM, Prime INFORMATION, PRIME/SNA, PRIMELINK, PRIMENET, PRIMEWAY, PRIMIX, PRISAM, PST 100, PT25, PT45, PT65, PT200, PW153, PW200, PW250, RINGNET, SIMPLE, 50 Series, 400, 750, 850, 2250, 2350, 2450, 2550, 2650, 2655, 2755, 9650, 9655, 9750, 9755, 9950, 9955, and 9955II are trademarks of Prime Computer, Inc.

PRINTING HISTORY

First Edition (DOC9473-11A) January 1985
Update 1 (UPD9473-11A) October 1985
Update 2 (UPD9473-12A) February 1986
Update 3 (UPD9473-13A) April 1986
Second Edition (DOC9473-21A) August 1987

CREDITS

Editorial: Thelma Henner
Project Support: The CPU Group
Illustration: Mingling Chang and Anna Spoerri
Document Preparation: Mary Mixon and Kathy Normington
Production: Judy Gordon

HOW TO ORDER TECHNICAL DOCUMENTS

To order copies of documents, or to obtain a catalog and price list:

United States Customers

Call Prime Telemarketing,
toll free, at 1-800-343-2533,
Monday through Friday,
8:30 a.m. to 5:00 p.m. (EST).

International

Contact your local Prime
subsidiary or distributor.

CUSTOMER SUPPORT

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (within Massachusetts)	1-800-541-8888 (within Alaska)
1-800-343-2320 (within other states)	1-800-651-1313 (within Hawaii)

For other locations, contact your Prime representative.

SURVEYS AND CORRESPONDENCE

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

Contents

ABOUT THIS BOOK	ix
1 SYSTEM OVERVIEW	
Single-stream Architecture	1-2
Instruction Pipeline Use	1-7
Special Features of the 6350	1-10
2 PHYSICAL AND VIRTUAL MEMORY	
Physical Memory	2-2
Virtual Memory	2-5
Summary	2-8
3 ADDRESSING	
Units	3-1
Components of a Virtual Address	3-2
Components of an Instruction	3-5
Forming an Address	3-8
Addressing Modes	3-10
Summary of Addressing Modes	3-12
Address Traps	3-31
Summary	3-35
4 MEMORY MANAGEMENT	
The Virtual Address	4-1
Memory Management Data Structures	4-3
Accessing the STLB and Cache	4-19
Address Translation	4-26
Summary	4-31
5 CONTROL INFORMATION AND RESTRICTED INSTRUCTIONS	
Other System Data Structures	5-1
Restricted Instructions	5-11
Summary	5-12

6	DATATYPES	
	Fixed-point Data	6-1
	Floating-point Numbers	6-19
	Decimal Data	6-32
	Character Strings	6-38
	Queues	6-41
	Summary of Datatypes and Applicable Instructions	6-46
	Summary	6-51
7	ALTERING SEQUENTIAL FLOW	
	Branch and Skip Instructions	7-1
	Jump Instructions	7-6
	Summary	7-7
8	STACKS AND PROCEDURE CALLS	
	Definition of Terms	8-1
	Stacks and Stack Management	8-2
	Entry Control Blocks	8-5
	Indirect Pointers	8-6
	Gate Access	8-7
	Making a Procedure Call	8-7
	The ARGT Instruction	8-14
	The PRTN Instruction	8-15
	Programming Notes	8-15
9	PROCESS EXCHANGE	
	Introduction	9-1
	Elements of the PXM	9-1
	Process Control Blocks	9-2
	Ready List	9-2
	Wait Lists	9-7
	PXM Instructions	9-9
	Dispatcher	9-16
	Register Files	9-16
	Process Interval Timer	9-25
	Dispatcher Operation	9-27
	Fetch Cycle Traps	9-30
	Summary	9-30
10	INTERRUPTS, FAULTS, CHECKS, AND TRAPS	
	Breaks	10-1
	Interrupts	10-3
	Faults	10-6
	Checks	10-18
	Traps	10-37
	Interval Clock	10-46
	Summary	10-47

11 INPUT-OUTPUT

Programmed I/O	11-2
DMx	11-10
DMA	11-16
DMC	11-19
DMT	11-20
DMQ	11-21
DMx Address Formation	11-21

APPENDIXES

A POWER-UP	A-1
B EARLIER PROCESSORS	B-1
System Overview	B-2
Physical and Virtual Memory	B-5
Addressing	B-6
Memory Management	B-8
Control Information and	
Restricted Instructions	B-11
Data Types	B-11
Altering Sequential Flow	B-16
Stacks and Procedure Calls	B-17
Process Exchange	B-17
Interrupts, Checks, Faults,	
and Traps	B-19
Input-Output	B-27
C PROCESS EXCHANGE ON THE 850	C-1
Instruction Stream Units	C-1
850 Process Exchange Elements	C-2
Dispatcher Operation	C-11
D INSTRUCTION SUMMARY CHARTS	D-1
E 2455 ARCHITECTURE	E-1
INDEX	X-1

About This Book

Prime's 50 Series™ family is a sophisticated group of totally compatible supermini computers. Its members are the Prime:

6350™	9955 II™	9955™	9950™
9755™	9750™	9655™	9650™
2755™	2655™	2550™	2450™
2350™	2250™	850™	750™
650™	550-II™	550™	500™
450™	I450™	400™	350™
250-II™	250™	150™	

The 50 Series systems embody an advanced 32-bit architecture that grants the user the ability to perform complex tasks efficiently and quickly. This document describes the 50 Series architecture from a functional point of view.

NOTES TO THE READER

Groups of people will find this document useful: engineers, programmers, designers, and technicians. To read this book, you should have a basic understanding of computers, but not necessarily of Prime computers. Prime stresses a high degree of compatibility across its product line; therefore, you can apply much of the information contained in this book to other Prime machines, as well as to the 50 Series machines.

ORGANIZATION OF THIS GUIDE

Because this guide stresses the functional aspects of the 50 Series processors, the topics are organized according to function. Chapter 1 presents a general overview. Chapters 2 through 11 each describe one aspect of the system, beginning with memory configuration and addressing and ending with the I/O system. Each chapter builds on the information contained in the previous one. Chapters 1 through 11 may be summarized as follows:

- Chapter 1 gives an overview of the 50 Series systems.
- Chapter 2 presents the configuration of the 50 Series physical and virtual memory.
- Chapter 3 discusses virtual addressing, modes and formats, and address traps.
- Chapter 4 describes memory management and its data structures.
- Chapter 5 gives the control data structures and restricted instructions.
- Chapter 6 specifies the datatypes supported on the 50 Series systems.
- Chapter 7 presents the branch, skip, and jump instructions.
- Chapter 8 defines procedure calls, the stack, and argument transfers.
- Chapter 9 describes single-stream process exchange and its data structures.
- Chapter 10 deals with interrupts, faults, checks, and traps.
- Chapter 11 discusses the I/O system (DMA, DMC, DMT, and DMQ).

Throughout these chapters are lists of Prime assembly language instructions that pertain to the topics under discussion. These lists briefly define the instructions' actions and show how they relate to the topics.

Appendix A discusses system power-up and the initialization of registers.

Appendix B presents the characteristics of the following earlier processors: 2250, 850, 750, 650, 550-II, 450/550, 500, I450, 400, 350, 250-II, 250, and 150.

Appendix C describes process exchange on the 850, a processor with dual-stream architecture.

Appendix D contains instruction summaries for all modes.

Appendix E describes the system architecture for the 2455.

PRIME DOCUMENTATION CONVENTIONS

The following conventions are used in command formats, statement formats, and in examples throughout this document. Examples illustrate the uses of these commands and statements in typical applications.

<u>Convention</u>	<u>Explanation</u>	<u>Example</u>
UPPERCASE	In command formats, words in uppercase indicate the names of commands, options, statements, and keywords. Enter them in uppercase.	CRL
lowercase	In command formats, words in lowercase indicate variables for which you must substitute a suitable value.	LDA address
Apostrophe	An apostrophe preceding a number indicates that the number is in octal.	'200

1

System Overview

The CPUs of all 50 Series systems share a common architecture and one operating system. This commonality is what makes the 50 Series a line of completely upward-compatible and downward-compatible systems. The implementation of the common architecture, however, is slightly different for each member, allowing the 50 Series systems to address a wide variety of user needs as well as to remain compatible. The first part of this chapter explores the single-stream CPU implemented on the 2350 to 9955 II. The second part of this chapter discusses special features of the 6350, the newest processor.

Note

The earlier processors 2250, 750, 650, 550-II, 550, 500, 450, I450, 400, 350, 250-II, 250, and 150 are also single-stream CPU processors. This chapter identifies where their single-stream implementation differs from the current processors. For a detailed discussion of these differences, refer to Appendix B. The 850, another earlier processor, is the only system with a dual-stream architecture that is discussed in Appendix B also.

SINGLE-STREAM ARCHITECTURE

The CPU can be divided into four major units. The first three of these are implemented on all single-stream members of the 50 Series family:

- Cache, STLB, and IOTLB
- Control store
- Execution unit
- Instruction unit

The instruction unit is a feature of the systems and serves as a mechanism to process instructions at a greater speed. Of the earlier processors, only the 750 and 850 have a fourth unit also, called the Instruction Preprocessor Unit and discussed in Appendix B.

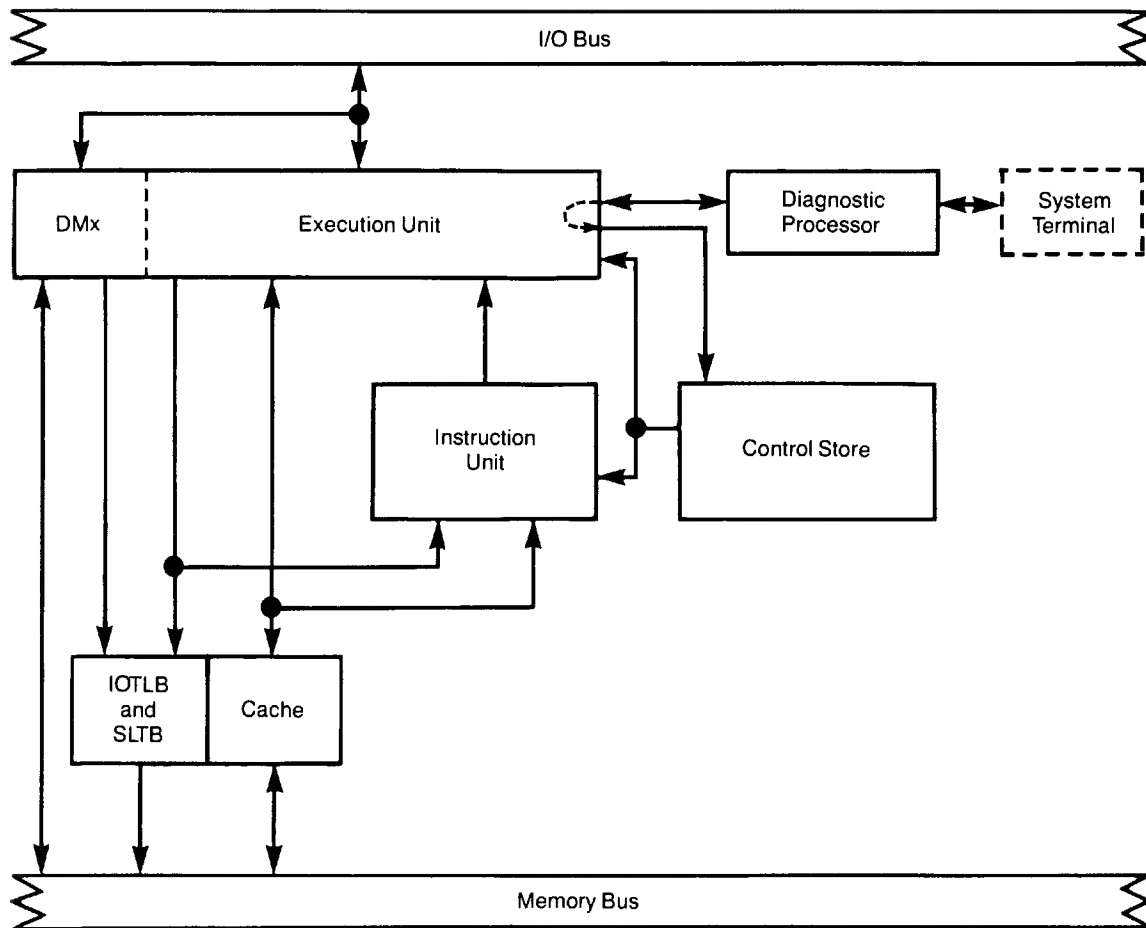
Figure 1-1 diagrams this architecture. This figure also shows the diagnostic processor which, among other functions, can load the control store and operate as the system terminal. The diagnostic processor also provides support for environmental sensors and the uninterruptable power supply. For the 6350, the diagnostic processor supports battery backup capability.

Cache, STLB, and IOTLB

The 50 Series uses a virtually addressed, write-through cache. Each cache entry contains the contents of 32 bits of recently accessed physical memory. Each entry also contains parity and valid bits as well as the physical page number that contains the 32 bits. (For the cache entry format on the earlier processors, see Appendix A.)

The 6350 has a two-set associative cache that is accessed in parallel to return two cache entries, each for the contents of 32 bits. Thus, two virtual addresses with the same cache index address can be used together without references to the one virtual address forcing the data for the other virtual address to be overwritten. This effectively eliminates two-way thrashing that could reduce performance.

If the contents of a specified location can be found in the cache, the system saves a great deal of time: it takes only 0.25 to 0.5 of the minimum instruction time to access the cache and get a cache hit, a vast improvement over the approximately 2 to 6 times the minimum instruction time needed to access physical memory. The time saved can be spent performing other operations rather than waiting for a memory reference to complete.



Block Diagram of Single-processor Architecture
Figure 1-1

To speed up the virtual to physical address translation, the STLB (Segmentation Table Lookaside Buffer) contains the results of the last translations: 1024 translations on the 6350; 512 translations on the 2350 to 2755, 9650, and 9655; and 128 translations on the 9750 to 9950.

The 6350 has a two-set associative STLB that is accessed in parallel to return two STLB entries. Thus, two virtual addresses with the same STLB entry address can be used together without references to the one forcing the mapping for the other virtual address to be overwritten. This effectively eliminates two-way thrashing that could reduce performance.

Because programs tend to reference the same set of locations during their execution, the system can perform a translation once, store the result in the STLB, and then have it for reference the next time the user specifies the same location. Because the STLB has a much faster access time than physical memory does, referencing it saves translation time as well as access time.

Mapped I/O allows the limited addressing range of DMx input/output transfers to address all of physical memory. It is especially useful when the processor is transferring several contiguous pages in virtual memory to physical locations that may not be contiguous. The IOTLB contains the information needed to map the transfer addresses to physical memory locations. The IOTLB, with the STLB, forms the virtual-to-physical address mapping hardware and contains 256 entries for the 6350, and 128 entries for the 2350 to 2755, 9650 to 9955 II.

See Chapter 4, MEMORY MANAGEMENT, for more information about cache, STLB, and address translation. See Chapter 11, INPUT-OUTPUT, for a description of the IOTLB.

The Control Store Unit

To speed up execution, the 50 Series systems implement many functions, such as procedure calls, in hardware and firmware. (Procedure calls are explained in Chapter 8.) The firmware that governs instruction execution is contained in the control store RAM: 80 Kbytes for the 6350; 50 Kbytes for the 9750 to 9955 II; 128 Kbytes for the 2755; and 64 Kbytes for the 2350 to 2655, 9650, and 9655.

The Execution Unit

This unit performs the computation required during instruction execution. Elements of the processor execution unit include:

- Integer arithmetic logic unit (ALU)
- Decimal ALU
- Floating point unit
- Register file

Figure 1-2 shows a diagram of the processor execution unit.

ALUs: The integer arithmetic logic unit (ALU) performs the desired operation on the user's two's complement data. In a similar fashion, the decimal ALU and the floating-point unit handle decimal and floating-point operations, respectively. These units can perform tests and checks as well as arithmetic operations.

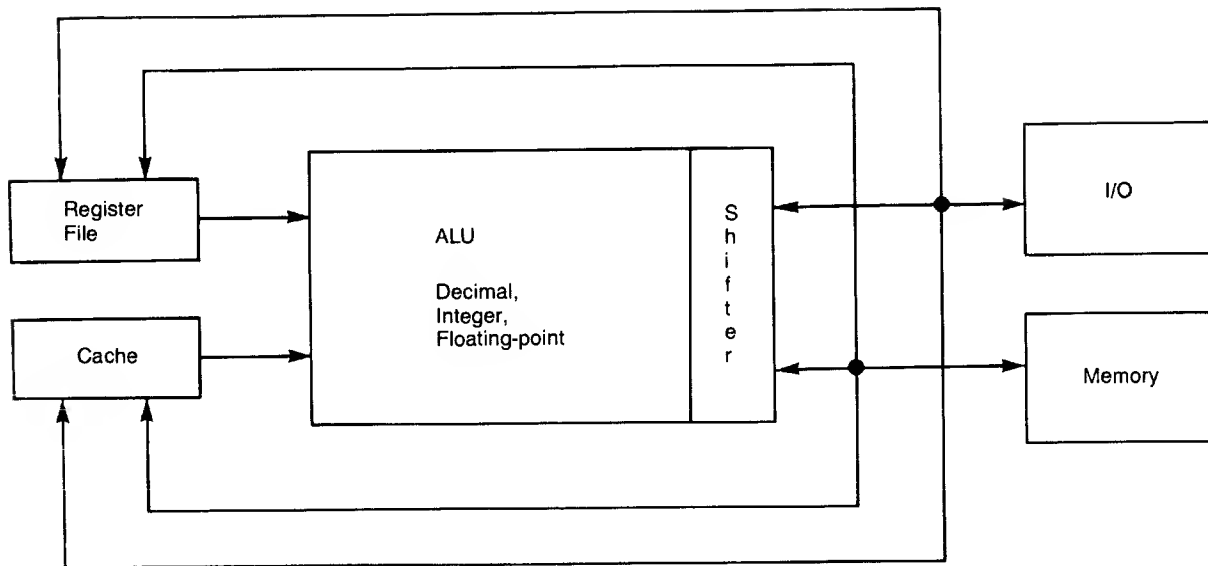
Register File: The register file contains up to eleven sets of registers, depending on the processor model. Each set contains 32 32-bit registers. There are three types of register sets: user, microcode scratch and system status, and DMA. User register sets contain information about a process and about the system as the process sees it. Specifically, user register sets contain information about the general registers a process can use, addresses of fault handlers, contents of system registers, and other useful information.

Direct memory registers contain direct memory access (DMA) channels to speed I/O operations as discussed in Chapter 11.

The 6350 and 9750 to 9955 II have eight register sets: four sets of user registers, two sets of microcode scratch and system status registers, one set of direct memory access registers, and one reserved.

The 2350 to 2755, 9650, and 9655 have eleven register sets: eight sets of user registers, three sets of microcode scratch and system status registers, and one set of direct memory access registers.

See Appendix B for the registers sets of the earlier processors listed on page 1-1.



Execution Unit
Figure 1-2

The Instruction Unit

The 2350 to 9955 II have an instruction unit designed to speed up execution by processing information about instructions before execution. The instructions are read from cache and decoded to provide the information necessary for effective address formation and for execution of the instruction.

INSTRUCTION PIPELINE USE

The 2350 to 9955 II use a pipeline to speed up instruction decoding and execution. The pipeline of the 6350 and 9750 to 9955 II has five stages. The pipeline of the 2350 to 2755, 9650, and 9655 has two phases. Both pipelines are discussed below.

The Five-Stage Pipeline

The 6350 and 9750 to 9955 II use a five-stage pipeline technique for executing instructions in parallel, thus speeding up instruction execution considerably. The execution of each instruction for this pipeline through the five stages is shown in Table 1-1. Each stage takes two beats to complete, where a beat is a certain constant of time. The beat rate is the minimal time interval that the processor requires to perform some useful task.

A processor using the five-stage pipeline executes instructions in parallel. This means that the processor does not have to complete the entire five-stage sequence for one instruction before it can begin executing the next. Rather, instructions are processed somewhat like cars in a factory assembly line. The cars travel past a number of specialized stations. At each station a specific operation takes place. Then the car moves on. After a certain length of time the next car arrives at the same station where the same operation occurs.

The five-stage pipeline processes instructions in a similar fashion. After every other beat, a new instruction arrives at a station, and that station's operation is performed on it.

Using the pipeline in this fashion, a processor executes Stage 1 of the first instruction. When it begins on Stage 2 (Beat 3) of the first instruction, that processor can also begin Stage 1 (Beat 1) of the second instruction. Likewise, when a processor begins Stage 2 (Beat 3) of the second instruction, it can also begin Stage 1 (Beat 1) of the third, and so on. This means that the pipeline can begin a new instruction every other beat.

Table 1-1
The Five-Stage Instruction Pipeline

Stage	Beat	Action
1	1	Send the contents of the lookahead program register to the memory address register.
1	2	Read the next instruction from the cache.
2	3	Start decoding the address of the next instruction.
2	4	Read the contents of the base and index registers.
3	5	Form the effective address and the control store address.
3	6	Send the contents of the effective address register to the memory address register and fetch the contents of the next microword.
4	7	Read the operand from the cache and register file.
4	8	Execution, phase 1 (ALU).
5	9	Execution, phase 2. (Transfer results to RS.)
5	10	Store the results of the operation.

The rate of instruction-flow through the pipeline is determined by the processor's use of system elements at each beat. As shown in Table 1-1, Beats 2 and 7 both use the cache, and Beats 7 and 10 both use the register file. When two instructions in the pipeline request the same element at the same time, a conflict occurs. Starting a new instruction every other beat minimizes this type of conflict.

When there are no conflicts in the pipeline, simple instructions complete execution every two beats. Some instructions, however, require more than two beats to complete execution. When this occurs, the pipeline holds up operations on the subsequent instructions until it has completed the extra operation for the first instruction. During the holdup, the processor still forms control store addresses and fetches microcode words, but it performs no prefetch or effective address calculations.

The Branch Cache and the Five-Stage Pipeline: The 6350 and 9750 to 9955 II use a memory called the branch cache to record and predict the target address for jump and branch instructions. The branch cache contains 256 to 1024 entries, depending on the processor model.

Because these processors execute instructions in parallel in their pipeline, they might begin to execute instructions down an incorrect path, following a branch, before they had determined the correct branch address. If this occurs, the processor must flush the pipeline of all instructions from the wrong branch path, and then must begin execution down the correct branch path. This sequence of steps causes a delay.

To minimize the chance of such an occurrence, the branch cache contains information about the branches that have previously occurred in the program. The processor uses this information to determine which branch was most recently taken for each conditional instruction. The processor then assumes that the same branch will be taken this time. If the prediction is wrong, the processor adds a new entry in the cache, specifying the correct branch for future use.

Flushing the Five-Stage Pipeline: If an instruction stores data into the stream of instructions that follows it, the five-stage pipeline may have to be flushed before further calculations take place. Store-instructions in S and R modes automatically flush the pipeline; therefore, no further actions are required and performance is reduced substantially. V mode and I mode store instructions, however, do not automatically flush the pipe. Either an E64V (V mode) or an E32I (I mode) instruction will perform the flush.

Prime systems are designed for pure procedure. All translator-generated code avoids storing into the instruction stream.

The Two-Phase Pipeline

The 2350 to 2755, 9650, and 9655 use a two-phase pipeline technique for decoding and executing instructions in parallel, thus speeding up instruction execution. While these processors perform the effective address formation and execution of one instruction, the next instruction is read from cache and decoded.

SPECIAL FEATURES OF THE 6350

Although the 6350 follows the general architecture of the 50 Series as shown in the previous discussions, it contains several features designed for outstanding performance in a multiuser environment.

Two-Set Associative STLB

The two-set associative STLB increases the likelihood that the physical translation of a virtual address is in the STLB. This lessens the chance that the slower virtual-to-physical address translation mechanism has to be used.

Two-Set Associative Cache

The two-set associative cache increases the probability that the cache will contain the correct data. This feature increases the likelihood that the physical translation of a virtual address is in the STLB. The combination of the two-set associative cache and the two-set associative STLB adds up to increased performance for the 6350.

10KH ECL Design

For swift execution of instructions, the 6350 uses 10KH ECL (emitter coupled logic). Memory parts using 10KH ECL are about twice as fast as those made of ECL at the same power level. Most of the logic is contained in semi-custom gate arrays.

Barrel Shifter

To speed up floating-point operations, the 6350 uses a barrel shifter. Moreover, this feature provides more power for manipulations performed in shift and rotate instructions.

Expanded I/O System

The 6350's I/O system has been expanded to speed up I/O performance and permit the parallel operation of a greater number of controllers. This feature is achieved through the use of new DMx operations and four I/O segments (0 to 3).

The 6350's new DMx operations are extended DMA, 32-bit single transfer DMA, 32-bit burst mode DMA, and 16-bit burst DMT.

In extended DMA, the DMA control words can be located anywhere in the I/O segments in memory, not just in the DMA register file, as long as the control word is 32-bit aligned. Single 32-bit DMA transfers 32 bits at a time instead of 16, and 32-bit burst mode transfers four 32-bit quantities at a time rather than four 16-bit ones.

DMT has been expanded on the 6350 to allow 16-bit burst DMT, whereby the CPU receives a main memory address and then reads or writes four 16-bit quantities at a time rather than just one.

Ambient Temperature Environmental Sensor

The 6350 now has an ambient temperature environmental sensor that detects when the air surrounding the processor has exceeded a certain temperature. Chapter 10 discusses this in further detail.

Battery Backup Capability

In the event of a power failure, the 6350 has a battery backup capability that keeps powered the memories, maintenance processor, and memory refresh logic of the CPU.

2

Physical and Virtual Memory

The 50 Series processors are virtual memory systems. This means that a very large, protected, virtual address space is available to each user who is logged onto the system. This virtual address space is supported by a much smaller physical address space invisible to the user.

Virtual memory has several advantages. To the user logged onto the system, there appears to be an address space of almost unlimited size, which can support very large applications without using overlays. This address space is protected against unauthorized accesses in hardware. To the system owner, a virtual memory scheme provides the ease of use of a large memory at the cost of a much smaller amount of hardware.

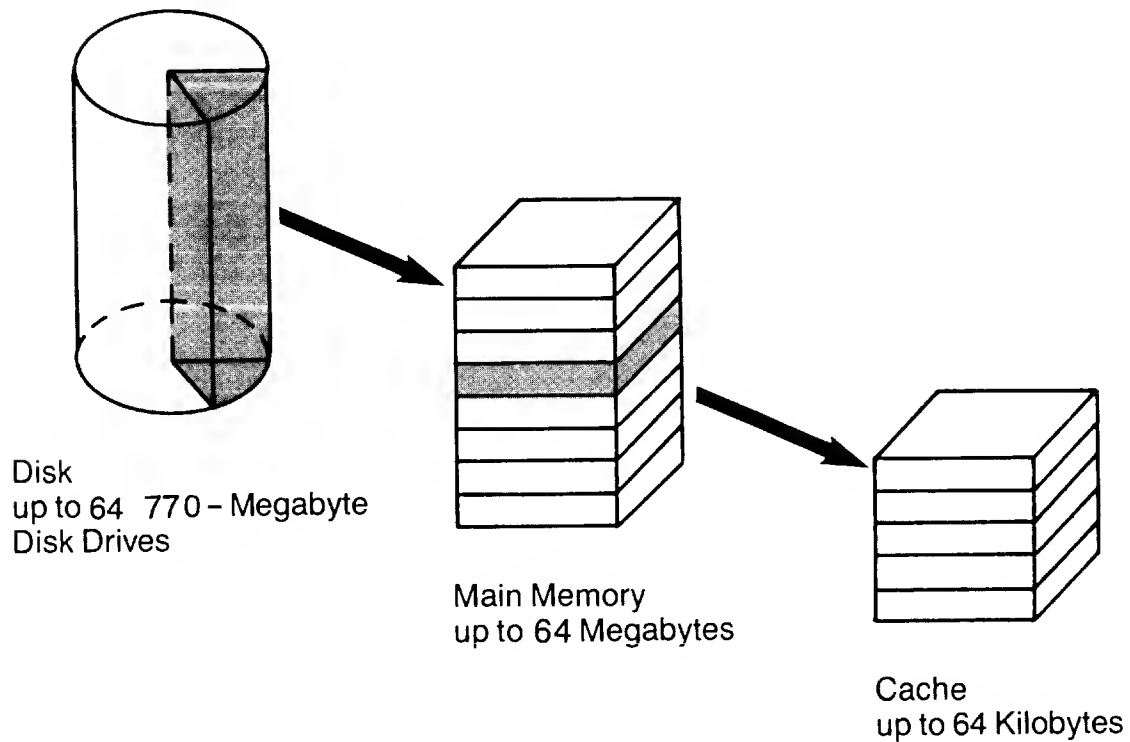
The three key parts to a virtual memory scheme are physical memory, virtual memory, and a manager to control the virtual memory scheme. The manager is the operating system, PRIMOS, and its attendant hardware and firmware support. This chapter describes the characteristics of the 50 Series physical and virtual memory, and shows how PRIMOS coordinates the 50 Series virtual memory scheme. It also describes some of the hardware protection mechanisms implemented in the 50 Series virtual memory.

PHYSICAL MEMORY

Physical memory encompasses all hardware parts of the system used to store large blocks of information. There are three types of physical memory:

- Cache
- Main memory
- Disk

Figure 2-1 shows the relationship between the three elements of physical memory.



Elements of Physical Memory
Figure 2-1

Cache

The cache is a data buffer that stores copies of the information contained in the most frequently referenced memory locations. Its size varies from system to system as shown in Table 2-1. During program execution, this buffer is used to speed up memory references.

Consider the following. Since cache is a form of very high speed memory, it takes only 0.25 to 0.5 of a minimum instruction time to access data stored there. In contrast, it takes about 2 to 5 times a minimum instruction time to access data stored in main memory. This difference in access times makes it very advantageous to access cache whenever possible.

Three factors determine how often the cache contains the correct data (known as the cache hit rate):

- The size of the cache (16 to 64 Kbytes)
- The organization of the cache (two-set associative or one-set)
- The information fetch rate (block size) of 32 to 64 bits, depending on the system and the amount of memory interleaving
- Locality of reference (the tendency of a program to execute within a small part of itself at any time)

The 50 Series cache hit rate varies from system to system. See Table 2-1 for details.

Table 2-1
Cache Sizes and Hit Rates*

System	No. Sets	Size Per Set	Total Size	Rate
2350 to 2655 and 9650 to 9950	1	16 Kbytes	16 Kbytes	95%
2755, 9955, and 9955 II	1	64 Kbytes	64 Kbytes	>98%
6350	2	16 Kbytes	32 Kbytes	>98%

*For earlier systems (listed on page 1-1), see Appendix B.

Main Memory

Packaged onto printed circuit boards, the main memory uses dynamic random access storage devices for data retention. All memory incorporates error detection and correction techniques and the capability of performing two-way interleaving.

Error detection and correction allows the memory to remain functional and to output correct data when a single bit in a 16-bit or 32-bit quantity (depending on processor model) has become faulty. This type of error is referred to as an ECCC. If more than a single bit in a single 16-bit quantity is in error, the fault is uncorrectable, an ECCU. All two-bit errors are detected as well as many multi-bit errors.

Interleaving effectively decreases the memory cycle time, increases memory accessibility, and allows more efficient use of the I/O bus.

There are two types of memory for all 50 series processors: the array card driven by a memory controller for the 6350 and 9750 to 9955 II; and the standalone memory subsystem for all other 50 Series processors.

The memory array cards used on the 6350 and 9750 to 9955 II require a memory control unit to supply commands, error detection and correction, and all interaction to and from the central processing unit. The 8-megabyte board has a 64-bit-wide storage capability that interacts directly with the memory bus. The total main memory capacity of the 6350 and 9750 to 9955 II is as follows.

9750 and 9755:	12 megabytes
9950 and 9955:	16 megabytes
9955 II:	32 megabytes
6350:	64 megabytes

Each standalone memory board used by the 2350 to 2755, 9650, and 9655 has a memory capacity of 2 or 4 megabytes to provide a maximum storage capacity as follows:

2350 to 2655:	8 megabytes
9650 and 9655:	8 megabytes
2755:	16 megabytes

The board itself has a 32-bit-wide storage capability that interacts directly with the memory bus.

Appendix B contains a description of the standalone memory subsystem for the earlier processors listed on page 1-1.

Disk

Disks provide storage for all virtual memory. With the proper access rights, the system or user can access this information. When the disk is accessed, a copy of the information is moved from disk to main memory.

VIRTUAL MEMORY

Virtual memory is divided into units called segments that contain up to 128 Kbytes each. Segments are virtual units, not physical ones, that aid the user and the system in organizing their virtual address spaces. For example, the user can organize program code in one segment and program data in a second one. Segments make it possible to allow extra room in a program for variable length data structures, such as arrays whose dimensions can change each time the program runs. Segments also allow the user to build modular programs, one module to a segment. PRIMOS uses segments similarly to organize its own code into modules.

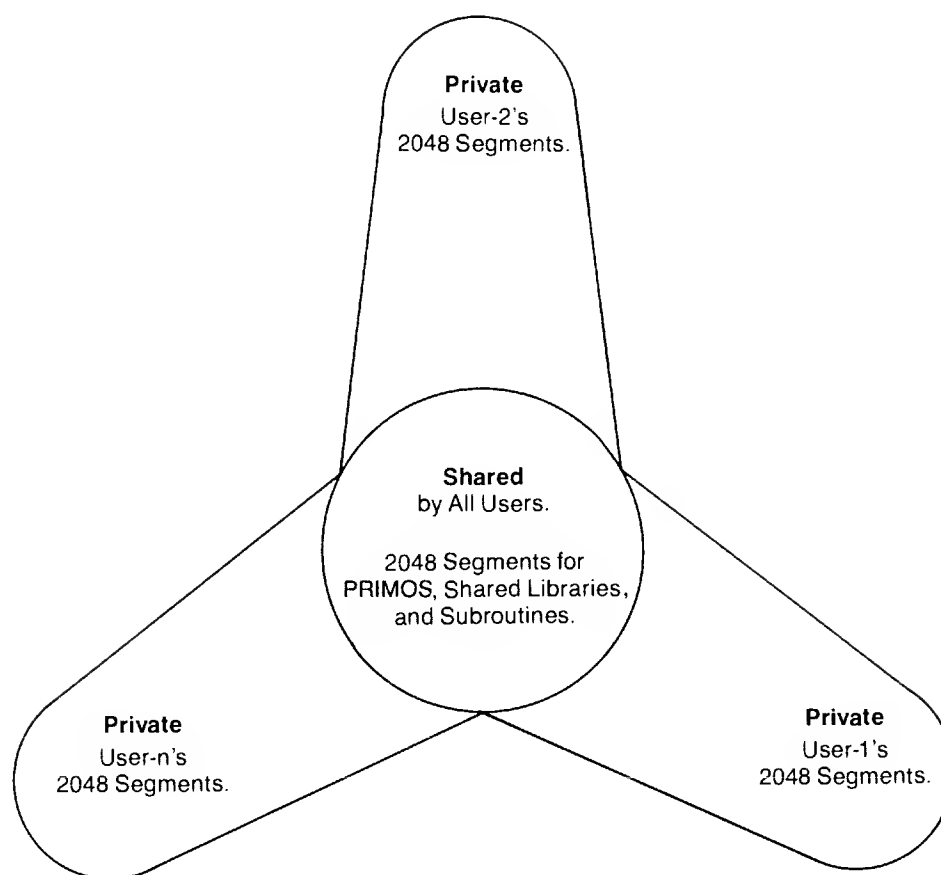
The virtual address space of each user contains 4096 segments. These are subdivided into four groups of 1024 segments each. The segments are subdivided to make address translation and segment sharing easier. (See the next section and Chapter 4, MEMORY MANAGEMENT.)

Shared and Unshared Segments

In the Prime virtual memory scheme (diagrammed in Figure 2-2), each user address space of 4096 segments is divided into shared and unshared space. The first 2048 segments are shared with all other users. This allows the operating system, shared libraries, and shared subsystems to be seen by all users. This means that if two users reference segment 2000, they are specifying the same location.

The second 2048 segments are private, containing information unique to each user. This means that if two users reference segment 4000, they are specifying completely different locations.

This arrangement of shared and unshared segments means that there is no possibility of one user's private space conflicting with that of another user. It also means that only one copy of PRIMOS and the shared system software need be maintained, and thus reduces memory use. Moreover, it means that PRIMOS is embedded in the virtual address space of each user and is directly accessible via a normal procedure call. (See Chapter 8, STACKS AND PROCEDURE CALLS.) No interrupts, special supervisor calls, or system traps are necessary when the user accesses PRIMOS or any utility, library, or subsystem residing in shared space.



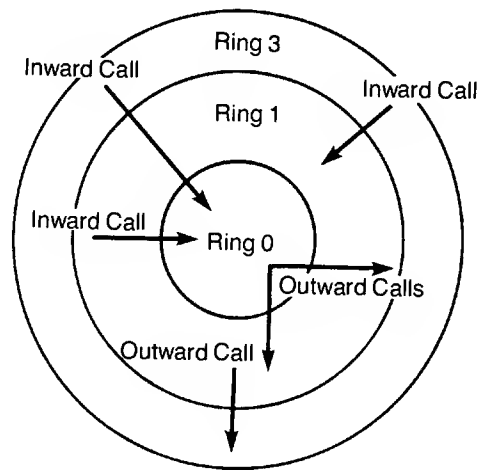
50 Series Virtual Memory Space
Figure 2-2

Protection Rings

Designating shared and unshared segments is not the only form of protection available to the 50 Series virtual memory. Three hardware implemented rings provide a simple, unbreakable form of security that checks each memory reference for its right to access the specified part of memory.

The rings represent levels of privilege, and are diagrammed in Figure 2-3. All executing procedures run with a given ring number. This ring value represents the rights, or privilege, of access in virtual memory. A process running under Ring 3 has the most restrictions, while a process running under Ring 0 has no restrictions.

Code that is executed under Ring 0 privilege must also have the greatest protection to prevent accidental or intentional misuse. The protection rings serve to provide this kind of protection so that a Ring 3 program is restricted from reading, writing, and/or executing Ring 0 data or code. Access is only provided to Ring 0 operating system routines through the use of special gates set up for that purpose.



Protection Rings
Figure 2-3

A Ring 3 program can therefore make a gated call to a Ring 0 routine. This is referred to as an inward call. Procedures that require greater access than is provided under Ring 3, but not as much as granted under Ring 0, operate under Ring 1 protection. Ring 1 procedures can also perform inward calls to Ring 0 procedures. See Chapter 8 for more information on procedure calls and gates.

See Chapter 4, MEMORY MANAGEMENT, for information about how rings govern the virtual-to-physical address translation to prevent invalid accesses.

Segmentation Table Lookaside Buffer

Virtual memory has its counterpart of the cache, the STLB. The system uses this buffer with the cache to reduce the time needed to access information. Where a cache entry contains information about a recently accessed physical memory location, an STLB entry contains the information the system needs to find the physical location from the virtual address the user specified. Each entry also specifies the protection attributes associated with the location. Chapter 4 describes more about how the STLB is used.

SUMMARY

This chapter describes the configuration of the 50 Series physical and virtual memories. Chapter 3, ADDRESSING, shows how to form a virtual address that references a location within the virtual address space. Chapter 4, MEMORY MANAGEMENT, shows how the 50 Series systems use the virtual address and the virtual-to-physical address translation process to integrate virtual and physical memory.

3

Addressing

The 50 Series processors support several kinds of addressing: direct addressing, indexed addressing, indirect addressing, indirect indexed addressing, and general register relative (GRR) addressing (unavailable for the earlier processors listed on page 1-1). They also support several modes of addressing, each with its own uses and benefits. This chapter:

- Provides an overview of virtual addressing and of effective address calculation.
- Explains how effective address calculation is done for each type of addressing, and what registers are involved.
- Explains the various modes of addressing.
- Provides summaries of instruction forms for each type of addressing in each mode.

UNITS

The basic units of information are bits, bytes, halfwords, and words. A byte contains eight bits. One halfword contains two bytes; the bits are labelled from 1 (most significant bit) to 16 (least significant bit). A word contains four bytes. The bits are labelled from 1 to 32.

Memory is measured in bytes. The 50 Series physical memory size can be up to 64 Mbytes; the virtual address space contains 512 Mbytes.

COMPONENTS OF A VIRTUAL ADDRESS

A virtual address refers to a unique location in a user's virtual address space. The location is characterized by three elements: a ring number, a segment number, and an offset within that segment. (All offsets are relative to the first location within a segment, and are expressed in units of halfwords.) The format of a virtual address is shown in Figure 3-1.

When an instruction makes a memory reference, it provides information from which the virtual address can be calculated. This is frequently referred to as calculating the effective address. Depending on the type of instruction, the information can be provided in several different formats, and the calculation done in various ways. This section explains the various ways in which the ring number, segment number, and offset can be specified. It also explains the use of the indirect bit. The section Forming an Address explains how each of the four types of addressing uses these components to calculate the effective address.

Ring Number

Ring numbers are found in the program counter, in the base register, within indirect addresses, and also in data blocks such as ECBs. When an effective address is calculated, the highest numbered ring referenced in any of these locations is chosen as the ring field for the effective address. (For more information on rings, and on the process of calculating ring numbers, see Chapter 4.)

Segment Number

The segment number is generally provided in one of four ways:

- If the instruction contains a base register field, the segment number is found in the specified base register.
- If the instruction does not contain a base register field, the segment number is found in the program counter.
- In indirect addressing, the segment number field contains the segment number.
- In I mode general register relative (GRR), bits 5 to 16 of the specified source register contain the segment number. (GRR is unavailable for the earlier processors listed on page 1-1.)

Base Registers: Four 32-bit base registers are available for use in address calculation:

- The procedure base register (PB)
- The stack base register (SB)
- The link base register (LB)
- The auxiliary base register (XB)

The format of these registers is shown in Figure 3-1.

1	2	3	4	5	16	17	32
0	RING	0	SEGMENT		OFFSET		

Bits	Name	Description
1	---	Must be 0. (See the F bit in the section on <u>Calculating Indirect Pointers</u> , in Chapter 8, for the explanation of this.)
2 to 3	Ring	Specifies the ring number.
4	---	Must be 0. (See the E bit in the section on <u>Calculating Indirect Pointers</u> , in Chapter 8, for the explanation of this.)
5 to 16	Segment	Specifies the segment number.
17 to 32	Offset	Specifies the offset value.

Format of Virtual Addresses and Base Registers
Figure 3-1

The PB contains the address of the currently active procedure. It is unique among the four base registers because its offset is always 0.

The program counter always contains a trusted copy of the segment number in the PB. Therefore, an instruction that contains no base register field uses the same segment number as one that specifies the PB.

SB contains the starting address of the stack for the currently active stack frame. LB contains the starting address of a save area for static variables, such as an entry control block. Because short instructions reference LB-based variables starting from '400, the value loaded into LB is usually '400 less than the start of the save area. References then add an extra '400 to their displacement. (See Chapter 8.) XB usually contains a temporary pointer, such as that to a FORTRAN common block. These three registers usually have nonzero offsets. Thus, they supply not only the segment number but also an offset address relative to that number.

Offset

The offset portion of an effective address is supplied by one or more of the following components:

- Displacement: a 16-bit number given explicitly within the instruction. In S, R, and V modes, the displacement can be 9 bits of the instruction that is added to or concatenated with the program counter.
- Base register: if the base register is SB, LB, or XB, it will contain an offset to be added to the displacement given within the instruction.
- Index register: if an index register is used, then the contents of that index register are to be added to whatever other offset has been calculated. When an I mode general register is used as an index register, only the contents of bits 1 to 16 are added to the offset.
- Indirect address: if indirect addressing is used, the indirect address contains the offset. Short form offsets are 16 bits. Long forms are 20 bits (bit pointers). Short form C language pointers are 17-bit offsets (byte pointers). (C language pointers are not available for the earlier processors listed on page 1-1.)
- Source register: if general register relative (GRR) is used, bits 17 to 32 of the source register will normally contain the offset. This is interpreted as the following C language pointer bits: the contents of bits 17 to 32 concatenated with the content of bit 4. (GRR is not available for the earlier processors listed on page 1-1.)

In summary, an offset can be calculated in any of the following ways:

- Displacement
- Displacement + offset from BR
- Displacement + index register (or source register low for GRR)
- Displacement + offset from BR + index register
- Indirect address
- Indirect address + index register

The instruction format tells the processor which method to use.

COMPONENTS OF AN INSTRUCTION

Instruction Format

Figure 3-2 diagrams a typical instruction format. Thus, it shows how all the fields described in this chapter fit together into a single instruction.

1	2	3	6	7	11	12	13	14	15	16	17	32
I	X	OP	11000	Y	OP	BR	DISP					

Bits	Mnem	Name	Description
1	I	Indirect bit	Specifies indirect addressing.
2	X	Index field	Specifies use of an index register.
3 to 6	OP	Opcode	Specifies the operation to perform.
7 to 11	---	----	Specifies instruction format.
12	Y	Index field	Specifies use of an index register.
13 to 14	OP	Opcode	Specifies the operation to perform.
15 to 16	BR	Base register	Specifies the base register to use.
17 to 32	DISP	Displacement	Specifies a 16-bit offset.

Format of a Typical Instruction (V Mode, Long)
Figure 3-2

The figure shown above explains the parts of a typical instruction. Instruction formats for all addressing modes, such as 64V short form or 32I, are provided later in this chapter.

Indirect Bit

An instruction may contain an indirect bit. If this bit is 1, it signifies that the address being calculated is an indirect address. If this bit is 0, the address is a direct address. (Indirect addresses are explained in the section Forming an Address, later in this chapter.)

Index Register Field

An instruction may specify two index registers by using the X and Y fields. Each of these fields is one bit long. These fields are encoded with the contents of the I field to specify the type of indexing to be performed. (See Table 3-4 for the encoding.) If an index register is specified, then the contents of that index register are added to whatever other offset has been calculated.

Base Register Field

The base register field of an instruction may contain one of the following four values:

<u>Value</u>	<u>Base Register</u>
00	PB (Procedure Base)
01	SB (Stack Base)
10	LB (Link Base)
11	XB (Auxiliary Base)

The value tells the processor which base register to check for the correct segment number (and, perhaps, offset).

Displacement

The displacement field contains a 16-bit number representing an offset within a segment. As the section on Offset explained, the value given by the displacement may either stand alone or have other values added to it to provide the actual offset for the effective address.

FORMING AN ADDRESS

The processor uses the contents of the fields in a memory reference instruction to select which of the four types of address formations to use:

- Direct
- Indexed
- Indirect
- Indirect indexed
- General register relative (for the 2350 to 9955 II only)

Direct Addressing

In direct addressing, the processor forms the effective address by adding the contents of the base register to the displacement.

Indexed Addressing

The processor adds the contents of the base register, index register, and displacement to produce the effective address.

S, R, and V mode instructions that contain 1101 in bits 3 to 6 cannot specify indexing. See the tables at the end of this chapter for specific information.

Indirect Addressing

Short Form Indirection: Depending on the addressing mode, indirect addressing takes one of two forms. In the first, the processor treats the displacement as the address of a location in the procedure segment. The processor uses the contents of the addressed location as the effective address. This is called short form, or 16-bit, indirection.

Some addressing modes allow more than one level of indirection. (See the 16S, 32S, and 32R sections at the end of this chapter.) In these cases, the processor uses the displacement as the address of some location in the address space. If this addressed location contains another indirect address, then the processor uses these contents as the address of another location in memory. This indirection chain is followed until one addressed location does not contain an indirect address; these contents are called the result of the chain. The processor uses the result of the chain as the effective address.

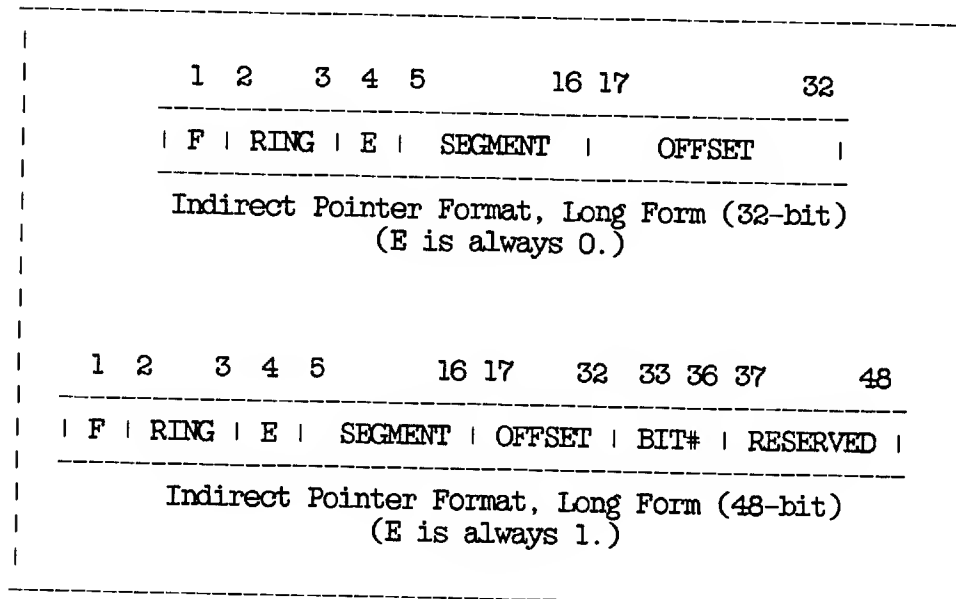
The tables at the end of this chapter specify the number of levels of indirection supported by each addressing mode.

Long Form Indirection: In long form indirect addressing, the instruction points to a location in memory that contains a 32-bit (or, more rarely, 48-bit) pointer. These long pointers contain not only addresses but also 2 or 3 fields that provide additional information.

Figure 3-3 shows the format of those pointers. The bits of special interest are the extension bit (or E bit), the fault bit (or F bit), and the bit number field.

The functions of these three fields are as follows:

F bit	If F = 1, a pointer fault is generated when this indirect address is used. (See Chapter 10 for information on pointer faults.)
E bit	If E = 0, the pointer is a 32-bit pointer. If E = 1, the pointer is a 48-bit pointer. (Throughout the rest of the chapter, discussions assume that the 32-bit format is being used.)
Bit number	Permits you to specify (or point to) a particular bit within an address offset.



Pointer Formats for Long Form Indirection
Figure 3-3

Indirect Indexed Addressing

This type of addressing takes one of two forms: indirect preindexed, or indirect postindexed.

When calculating a preindexed indirect address, the processor adds the value of the index register to the contents of the base register and displacement and uses the sum as an indirect address. It resolves any indirection chain and uses the result of the chain (or the indirect address itself, if there was no chain to follow) as the effective address.

When calculating a postindexed indirect address, the processor adds the contents of the base register and displacement and uses the result as an indirect address. It resolves any indirection chain, then adds the result of the chain (or the indirect address itself, if there was no chain to follow) to the contents of the specified index register to form the effective address.

General Register Relative Addressing

General register relative (GRR) is an addressing capability added to 32I mode that speeds up big array accesses and often gives the effect of using general registers as base registers. The segment number is formed from bits 5 to 16 of the specified source register. The offset is formed in GRR by adding the displacement to bits 17 to 32 of the specified source register. GRR is used by the I mode instructions AIP and LIP. (GRR is not available for the earlier processors listed on page 1-1.)

ADDRESSING MODES

The first part of this chapter describes several ways to specify an address with information contained within an instruction. Once the processor calculates the effective address, it can reference whatever information is contained in the location specified by the effective address. This section describes the ways to specify an address in an instruction and how the processor forms the effective address.

The 50 Series processors support four modes of addressing, each of which forms addresses differently. Depending on the program and personal preference, one or two of these modes may be more useful than another. The three most important modes are:

- V, or virtual
- I, or general register
- R, or relative

The fourth mode -- S, or sectored, mode -- is supported for historical reasons.

V Mode

V mode performs short and long operations and has a wide variety of registers to use. A short (16-bit) instruction in this mode can reference the first 256 locations of both the stack and link, as well as the 224 locations on either side of the current location in the procedure segment. A long (32-bit) V mode instruction can directly reference all locations in four segments. Indirect addressing can reference all locations in up to 4096 128-Kbyte segments.

I Mode

When referencing memory, I mode is similar to 32-bit V mode. The difference is that I mode short operations reference 8 32-bit general purpose registers for use as index registers, accumulators, counters, or the like. I mode long operations have the same referencing power as V mode long operations. They can also use immediate forms and five additional index registers. (This makes a total of 7 index registers that I mode long operations can use.) The index registers are specified by the source register field. General register 0, however, cannot be used for indexing.

General register relative (GRR) addressing is available only in I mode, and is used by the I mode instructions AIP and LIP. This form of addressing speeds up big array accesses and often gives the effect of using general registers as base registers. (GRR is not available for the earlier processors listed on page 1-1.)

The C language pointer is used by the I mode instructions ACP, CCP, DCP, ICP, LOC, SOC, and TCNP. The format of this pointer is the same as the indirect pointer, except that bit 4 is redefined as the B (byte) bit. When this bit contains 0, it indicates that bits 1 to 8 (the left byte) of an address contain the character to be used; when this bit contains 1, bits 9 to 16 (the right byte) of an address contain the character. A null pointer is represented by zeroes in bits 4 to 32. (The C language pointer and its instructions are not available for the earlier processors listed on page 1-1.)

Normal effective address formation uses either a base register, indirect pointer (IP) or a general register (for GRR addressing) as the source of the ring field, B bit, and segment number. The C language pointer is well defined for the IP and GRR form. When the base register is the source of the B bit, software depends on finding it reset to zero, pointing to the leftmost byte. While it is possible to set the E bit in a base register using 48-bit IPs to specify 32-bit addresses, this practice is not now done. Future implementations of V and I modes will force bit 4 to zero during effective address formation

when the source of the segment is a base register; otherwise it will copy bit 4.

R Mode

A sector is a block of 512 (1000 octal) contiguous memory locations. Sector 0 starts on location 0 and ends on location '777; Sector 1 begins on location '1000 and ends on location '1777; and so on.

An R mode instruction can reference any location in Sector 0, as well as a group of locations relative to the current value of the program counter. When the sector bit (S) in an R mode instruction is 0, the instruction can only reference locations in Sector 0. When S is 1, the instruction references locations relative to the current value of the program counter. The range of these relative locations is PC - '360 to PC + '377, inclusive.

An R mode instruction that specifies a location in the range PC - '361 to PC - '400, inclusive, selects a special addressing code, such as stack register. These special codes are explained in more detail in Tables 3-7 and 3-8.

S Mode

Like R mode instructions, S mode instructions contain a sector bit. When S is 0, references are to Sector 0. When S is 1, however, references are only to those locations within the sector containing the instruction.

Note that S mode is a holdover from early Prime machines that were based on the Honeywell 316 and 516 minicomputers. When operating in S mode, the 50 Series processors act exactly as these early machines do.

SUMMARY OF ADDRESSING MODES

The figures and tables in the rest of this chapter present summaries of each addressing mode. Table 3-1 is a list of the mnemonics used in these addressing mode summaries. Table 3-2 summarizes useful information about all the modes.

Table 3-1
Mnemonics Used in Summaries of Addressing Modes*

Mnem	Explanation	Mnem	Explanation
BR	Base register	REG	A location in the register file. See <u>Address Traps</u> .
CB	Class bit		
D	Displacement	S	Sector bit
DR	Destination register	SB	Stack base register
F	Fault bit	SP	Stack pointer
I	Indirect bit	SR	Source register
LB	Link base register	TM	Tag modifier
OP	Opcode	X	X index register
P	PC + 1	XB	Auxiliary base register
PB	Procedure base register**	XX	Opcode extension
PC	Program counter**	Y	Y index register

* An H appended to a register mnemonic refers to bits 1 to 16 of that register; an L so appended refers to bits 17 to 32.

** The PB segment number equals the PC segment number. The PB offset number is 0, but the PC offset number is the next instruction.

Table 3-2
Summary of Addressing Modes

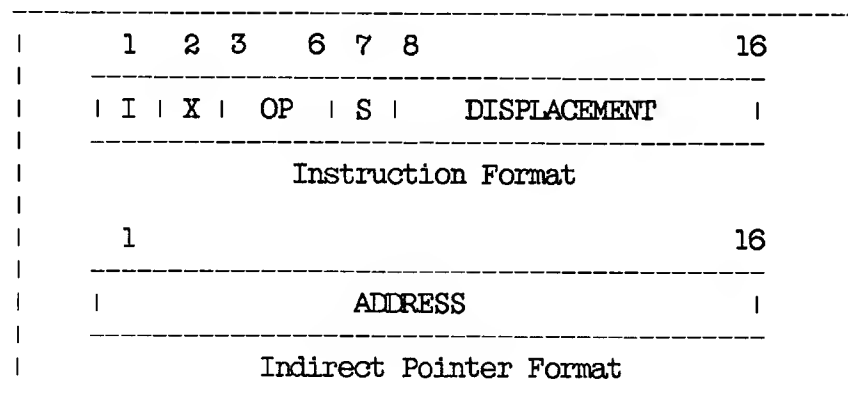
Mode	Address Length	Addressing Range	# Index Regs	Indirection Levels
16S direct	14 bits	1024 halfwords	One	
16S indirect	14 bits	16K halfwords	One	Multiple
32S direct	15 bits	1024 halfwords	One	
32S indirect	15 bits	32K halfwords	One	Multiple
32R direct	15 bits	1008 halfwords	One	
32R indirect	15 bits	32K halfwords	One	Multiple
64R direct	16 bits	1008 halfwords	One	
64R indirect	16 bits	64K halfwords	One	One
64V 16-bit instructions	16 bits	64K halfwords: +256 SB relative +256 LB relative +/-256 PC relative +512 PB absolute	One	One
64V 32-bit instructions	28 bits	4 segments*	Two	One
64V indirect	28 bits	4096 segments*	Two	One
32I all	28 bits	12 segments* with GRR**	Seven	One
32I indirect	28 bits	4096 segments*	Seven	One

* All segments contain 128 Kbytes.

** Four segments for the earlier processors listed on page 1-1 because they have no GRR capability.

64V Mode Short Form

Figure 3-4 and Table 3-3 display and explain 64V mode short form instructions.



64V Mode Formats, Short Form
Figure 3-4

Table 3-3
64V Mode Short Form Summary

I	X	S	Disp	Inst Type	Example	Form of EA
0	0	0	0-'7@	Direct	LDA ADR	REG
			'10-'377	Direct		SB+D
			'400-'777	Direct@@		LB+D
0	1	0	0-'7@	Indexed	LDA ADR,X	REG, if D+X<'7;@
			'10-'377	Indexed		SB+D+X, if D+X>'7@
			'400-'777	Indexed@@		SB+D+X
						LB+D+X
1	0	0	0-'7@	Indirect	LDA ADR,*	I(REG)
			'10-'777	Indirect		I(PB+D)
1	1	0	0-'7	Indirect, preindexed	LDA ADR,X*	I(REG), if D+X<'7;@
						I(PB+D+X),
						if D+X>'7@
			'10-'77	Indirect, preindexed	LDA ADR,X*	I(PB+D+X)
			'100-'777	Indirect, postindexed	LDA ADR,*1	I(PB+D)+X
0	0	1	'-340-' +377	Direct	LDA ADR	P+D
0	1	1	'-340-' +377	Indexed	LDA ADR,1	P+D+X
1	0	1	'-340-' +377	Indirect	LDA ADR,*	I(P+D)
1	1	1	'-340-' +377	Indirect, preindexed	LDA ADR,1*	I(P+D+X)

Notes to Table 3-3

@ This table assumes segmented mode (modals bit 14 = 1). For nonsegmented mode, the displacement range is 0 to '37, rather than 0 to '7. This means that the range '10 to '377 changes to '40 to '377 in nonsegmented mode. The range '400 to '777 remains unchanged.

@@ In these address forms, the displacement offsets the contents of LB by '400 (bit 8=1). To compensate for this, set the contents of LB to the current value of the link frame minus '400. For example, if the segment number in LB is '4002 and the offset number in the displacement is '177400, the offset of '400 gives the location of the link frame as segment number '4002, offset number 0.

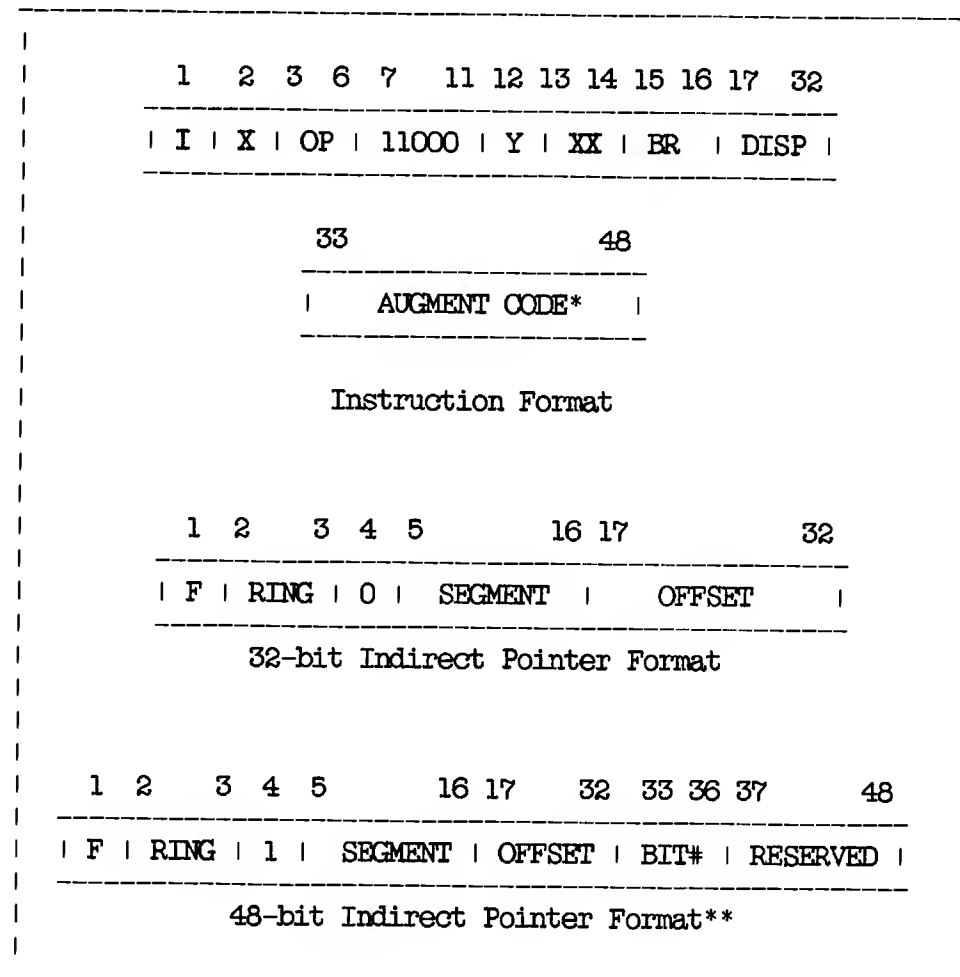
This mode allows one level of indexing, and one of indirection.

REG refers to a location in the register file. See Address Traps at the end of this chapter.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The effective address is formed as if bit 2 = 0.

64V Mode, Long Form and Indirect Form

Figure 3-5 and Table 3-4 display and explain 64V mode long and indirect form instructions.



* For quad operations only.

** This indirect format is used only by a few instructions; most use the 32-bit form.

64V Mode Formats, Long Form and Indirect Form
Figure 3-5

Table 3-4
64V Mode Long Form, Indirect Summary

I	X	Y	BR	Instruction Type	Example	Form of EA
0	0	0	00	Direct	LDA ADR	PB/D
			01			SB+D
			10			LB+D
			11			XB+D
0	0	1	00	Indexed by Y	LDA ADR,Y	PB/D+Y
			01			SB+D+Y
			10			LB+D+Y
			11			XB+D+Y
0	1	0	00	Indexed by X	LDA ADR,X	PB/D+X
			01			SB+D+X
			10			LB+D+X
			11			XB+D+X
0	1	1	00	Indirect	LDA ADR,*	I(PB/D)
			01			I(SB+D)
			10			I(LB+D)
			11			I(XB+D)
1	0	0	00	Preindexed by Y	LDA ADR,Y*	I(PB/D+Y)
			01			I(SB+D+Y)
			10			I(LB+D+Y)
			11			I(XB+D+Y)
1	0	1	00	Postindexed by Y	LDA ADR,*Y	I(PB/D)+Y
			01			I(SB+D)+Y
			10			I(LB+D)+Y
			11			I(XB+D)+Y
1	1	0	00	Preindexed by X	LDA ADR,X*	I(PB/D+X)
			01			I(SB+D+X)
			10			I(LB+D+X)
			11			I(XB+D+X)
1	1	1	00	Postindexed by X	LDA ADR,*X	I(PB/D)+X
			01			I(SB+D)+X
			10			I(LB+D)+X
			11			I(XB+D)+X

Notes to Table 3-4

The processor performs X and Y indexing and 32-bit word (inter-segment) indirection.

PB/D indicates that the displacement is relative to the origin of PB. PB specifies the segment number (the offset must be 0); the displacement specifies the offset.

All displacements are within the range 0 to '177777.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The effective address is formed as shown in Table 3-5. Bit 2, the X bit, is used as part of the opcode in these instructions.

Table 3-5
Address Formation for Nonindexing Instructions

I	X	Y	Instruction Type
0	0	0	Direct
0	0	1	Direct
0	1	0	Direct
0	1	1	Direct
1	0	0	I(A)
1	0	1	I(A)
1	1	0	I(A)
1	1	1	I(A)

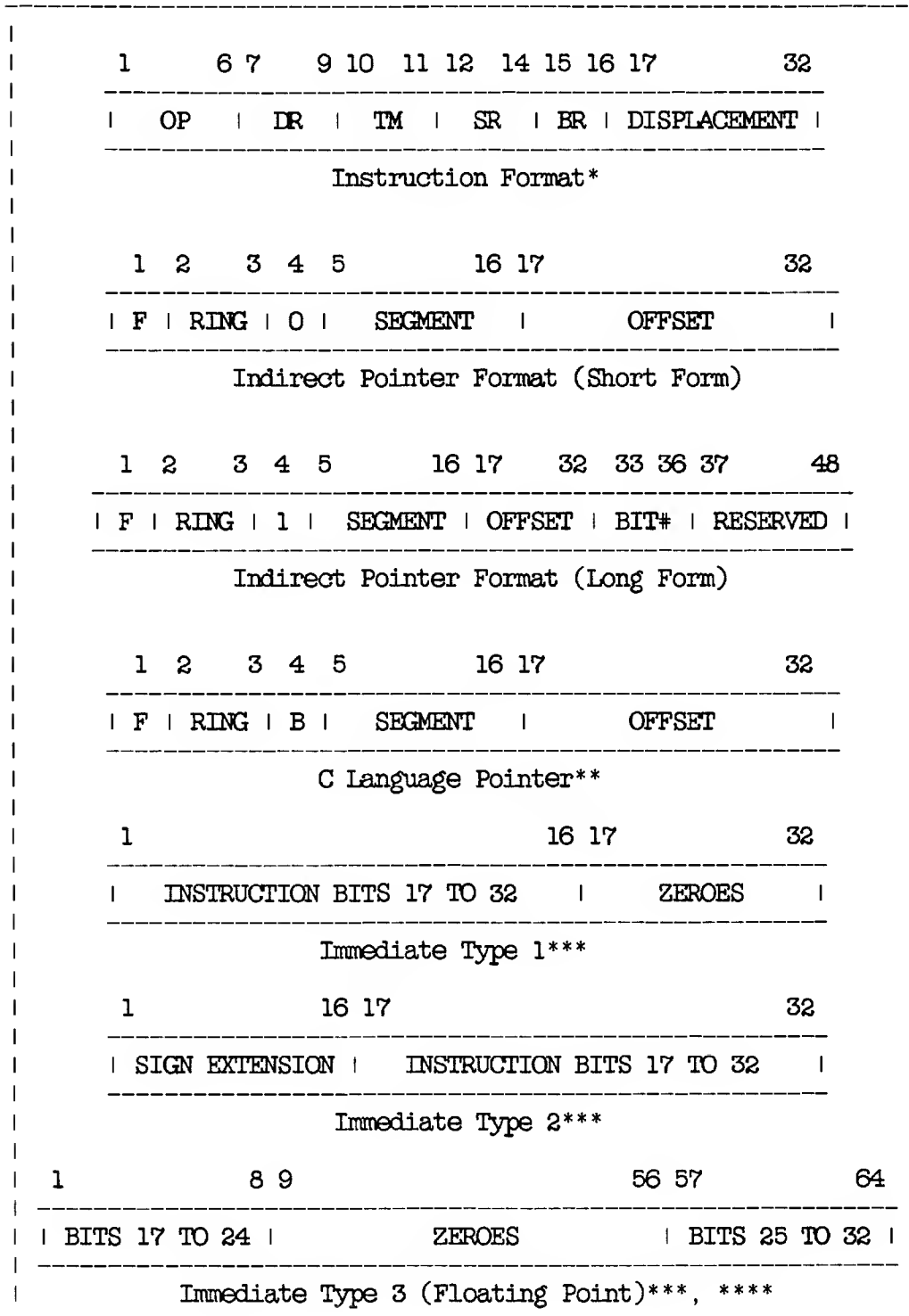
Notes to Table 3-5

For the earlier processors listed on page 1-1, see Appendix B for information on their address formation for nonindexing instructions.

The symbol A in Table 3-5 represents the value calculated from the base register (PB, SB, LB, or XB) and displacement in the instruction.

32I Mode

Figure 3-6 and Table 3-6 display and explain 32I mode instructions.



32I Mode Formats
Figure 3-6

Notes to Figure 3-6

- * TM is the tag modifier which, in combination with the SR and BR fields, specifies the instruction type.
- ** The C language pointer is not available for the earlier processors listed on page 1-1.
- *** The instruction specifies the immediate type to use. During instruction execution, the processor forms the immediate in the appropriate format and stores it internally for use in the operation as shown in Figure 3-6.
- **** Bits 1 to 8 of Immediate Type 3 are formed from I mode instruction bits 17 to 24; bits 57 to 64 from I mode instruction bits 25 to 32.

Table 3-6
32I Mode Summary

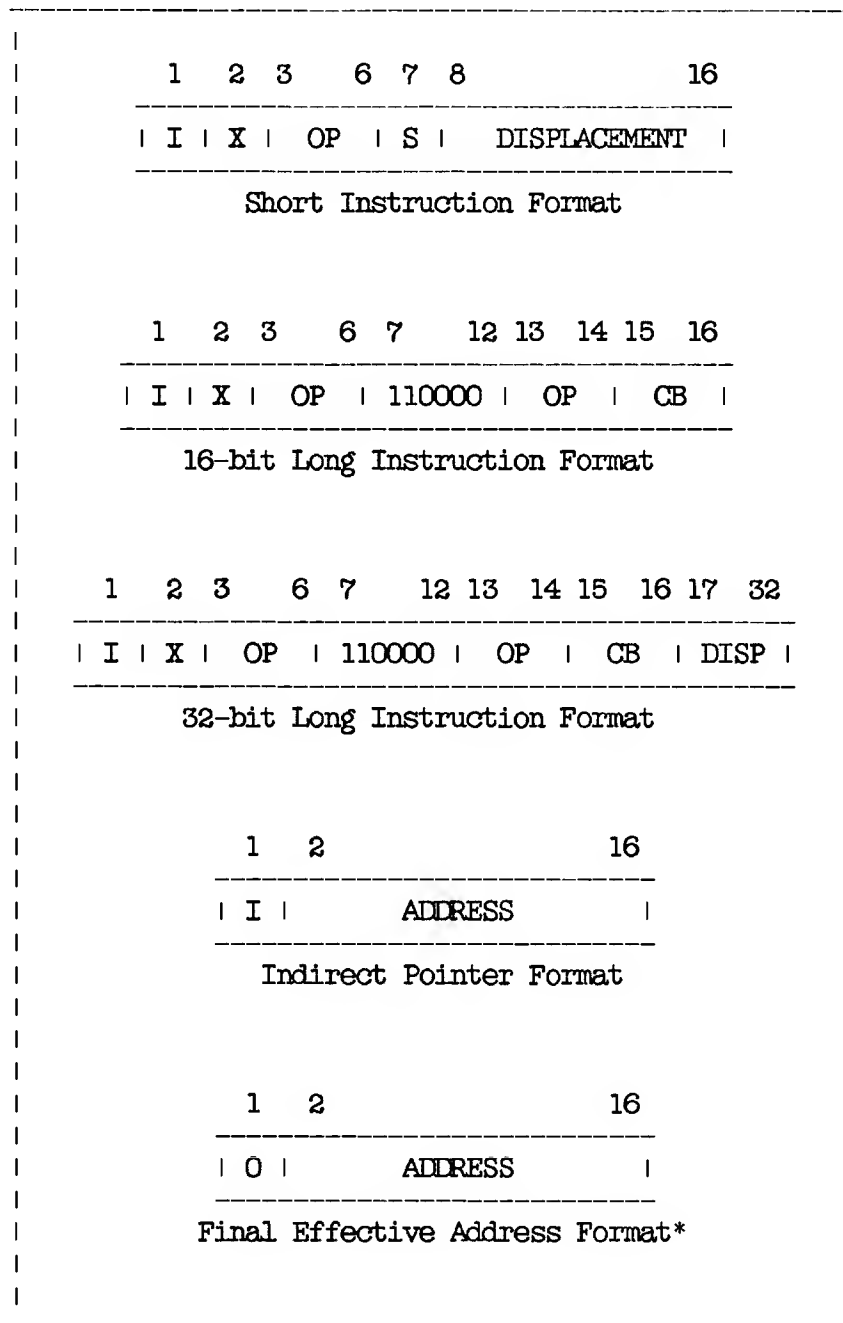
TM	SR	BR	Instruction Type	EA (Segment)	EA (Offset)
3	0	-	Indirect	I(5 to 16)	I(D+BR)
3	>0	-	Indirect postindexed	I(5 to 16)	(I(D+BR))+SRH
2	0	-	Indirect	I(5 to 16)	I(D+BR)
2	>0	-	Indirect preindexed	I(5 to 16)	I(D+BR+SRH)
1	0	-	Direct	BR(5 to 16)	D+BR
1	>0	-	Indexed	BR(5 to 16)	D+BR+SRH
0	0-7	0	Register-to-register	---	---
0	0	1	Immediate type 1	---	---
0	>0	1	Immediate type 2	---	---
0	0	2	Immediate type 3	---	---
0	1	2	Floating register source (FR0)	---	---
0	2	2	Undefined; generates UII (unimplemented instruction) fault	---	---
0	3	2	Floating register source (FR1)	---	---
0	4-7	2	Undefined; generates UII fault	---	---
0	0-7	3	General register relative (undefined for the earlier processors listed on page 1-1)	SR(5 to 16)	D+SRL

Note to Table 3-6

Displacements are within the range 0 to '177777, inclusive.

32R Mode

Figure 3-7 and Table 3-7 display and explain 32R mode instructions.



32R Mode Formats
Figure 3-7

Note to Figure 3-7

The final form of an effective address in 32R mode is only 15 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide. Multilevel indirection is a feature of 32R mode.

Table 3-7
32R Mode Summary

I	X	S	CB	Displacement	Instruction Type	Form of EA
0	0	0	--	0 to '777	Direct	O/D
0	1	0	--	0 to '777	Indexed	O/D+X
1	0	0	--	0 to '777	Indirect	I(O/D)
1	1	0	--	0 to '77	Indirect, preindexed	I(O/D+X)
1	1	0	--	'100 to '777	Indirect, postindexed	I(O/D)+X
0	0	1	--	'-360 to '+377	Direct	P+D
0	1	1	--	'-360 to '+377	Indexed	P+D+X
1	0	1	--	'-360 to '+377	Indirect	I(P+D)
1	1	1	--	'-360 to '+377	Indirect postindexed	I(P+D)+X
0	0	1	2	---	@Postincrement	SP
0	1	1	2	---	@Postincrement, indirect, postindexed	I(SP)+X
1	0	1	2	---	@Postincrement, indirect	I(SP)
0	0	1	3	---	#Predecrement	SP-1
0	1	1	3	---	#Predecrement, indirect, postindexed	I(SP-1)+X
1	0	1	3	---	#Predecrement, indirect	I(SP-1)
0	0	1	0	0 to '177777	*Direct, long reach	D
0	1	1	0	0 to '177777	*Indexed, long reach	D+X
1	0	1	0	0 to '177777	*Indirect, long reach	I(D)
1	1	1	0	0 to '177777	*Indirect, preindexed, long reach	I(D+X)
1	1	1	2	0 to '177777	*Indirect, postindexed, long reach	I(D)+X
0	0	1	1	0 to '177777	*Direct, stack relative	D+SP
0	1	1	1	0 to '177777	*Indexed, stack relative	D+SP+X
1	0	1	1	0 to '177777	*Indirect, stack relative	I(D+SP)
1	1	1	1	0 to '177777	*Indirect, preindexed stack relative	I(D+SP+X)
1	1	1	3	0 to '177777	*Indirect, postindexed stack relative	I(D+SP)+X

Notes to Table 3-7

- * These instruction types use the 32-bit long format shown in Figure 3-7.
- @ These instruction types use the 16-bit long format shown in Figure 3-7. They also increment the contents of SP by 1 during EA formation.
- # These instruction types use the 16-bit long format shown in Figure 3-7. They also decrement the contents of SP by 1 during EA formation.

For all instruction types listed above, address traps can occur when any part of the EA formation results in an address in the range 0 to '7 (segmented mode) or 0 to '37 (unsegmented mode). See the end of this chapter for more information.

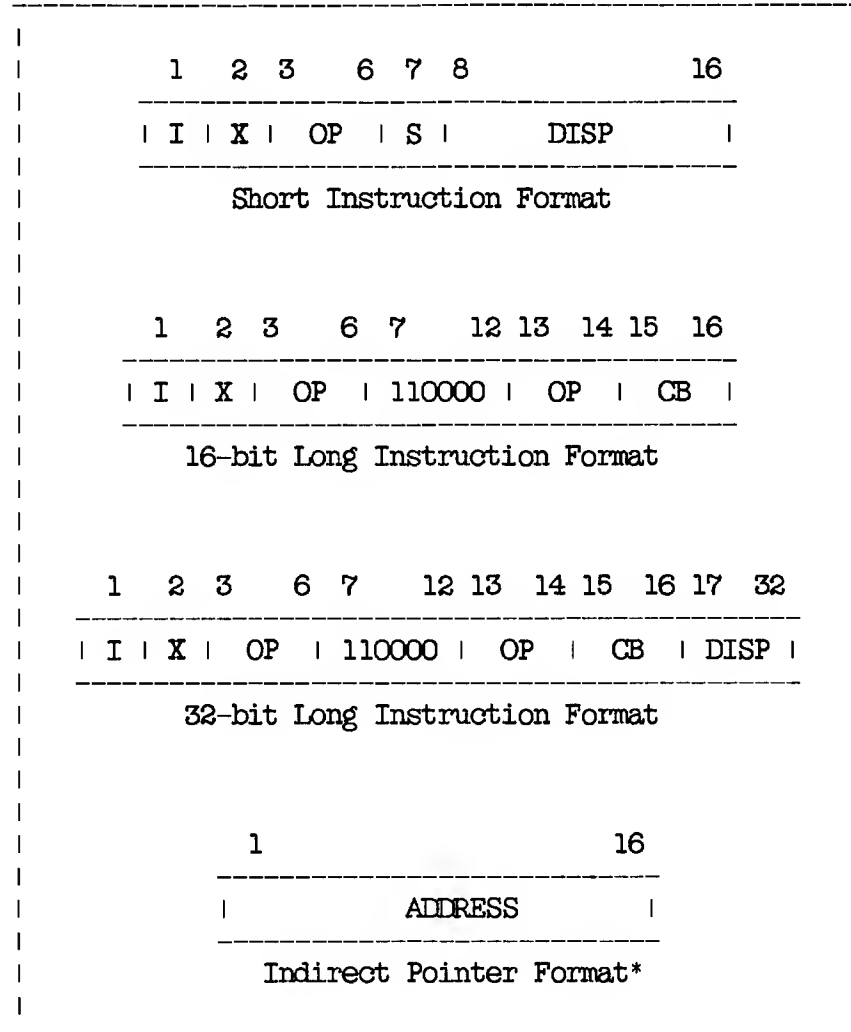
The processor performs one level of indexing and multiple levels of indirection.

O/D indicates that the displacement is within Sector 0.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The processor treats the X bit as a 0 to determine what addressing mode to use. For example, if one of these instructions specifies I, X, S, and CB as 0113, the processor interprets it as 0013.

64R Mode

Figure 3-8 and Table 3-8 display and explain 64R mode instructions.



*Only a single level of indirection is possible in 64R mode.

64R Mode Formats
Figure 3-8

Table 3-8
64R Mode Summary

I	X	S	CB	Displacement	Instruction Type	Form of EA
0	0	0	--	0 to '777	Direct	O/D
0	1	0	--	0 to '777	Indexed	O/D+X
1	0	0	--	0 to '777	Indirect	I(O/D)
1	1	0	--	0 to '77	Indirect, preindexed	I(O/D+X)
1	1	0	--	'100 to '777	Indirect, postindexed	I(O/D)+X
0	0	1	--	'-360 to '+377	Direct	P+D
0	1	1	--	'-360 to '+377	Indexed	P+D+X
1	0	1	--	'-360 to '+377	Indirect	I(P+D)
1	1	1	--	'-360 to '+377	Indirect postindexed	I(P+D)+X
0	0	1	2	---	@Postincrement	SP
0	1	1	2	---	@Postincrement, indirect, postindexed	I(SP)+X
1	0	1	2	----	@Postincrement, indirect	I(SP)
0	0	1	3	---	#Predecrement	SP-1
0	1	1	3	----	#Predecrement, indirect, postindexed	I(SP-1)+X
1	0	1	3	----	#Predecrement, indirect	I(SP-1)
0	0	1	0	0 to '1777777	*Direct, long reach	D
0	1	1	0	0 to '1777777	*Indexed, long reach	D+X
1	0	1	0	0 to '1777777	*Indirect, long reach	I(D)
1	1	1	0	0 to '1777777	*Indirect, preindexed, long reach	I(D+X)
1	1	1	2	0 to '1777777	*Indirect, postindexed, long reach	I(D)+X
0	0	1	1	0 to '1777777	*Direct, stack relative	D+SP
0	1	1	1	0 to '1777777	*Indexed, stack relative	D+SP+X
1	0	1	1	0 to '1777777	*Indirect, stack relative	I(D+SP)
1	1	1	1	0 to '1777777	*Indirect, preindexed stack relative	I(D+SP+X)
1	1	1	3	0 to '1777777	*Indirect, postindexed stack relative	I(D+SP)+X

Notes to Table 3-8

For all the instruction types listed in Table 3-7, address traps can occur when any part of the EA formation results in an address in the range 0 to '7 (segmented mode) or 0 to '37 (unsegmented mode). See the end of this chapter for more information.

* These instruction types use the 32-bit long format shown in Figure 3-8.

@ These instruction types use the 16-bit long format shown in Figure 3-8. They also increment the contents of SP by 1 during EA formation.

These instruction types use the 16-bit long format shown in Figure 3-8. They also decrement the contents of SP by 1 during EA formation.

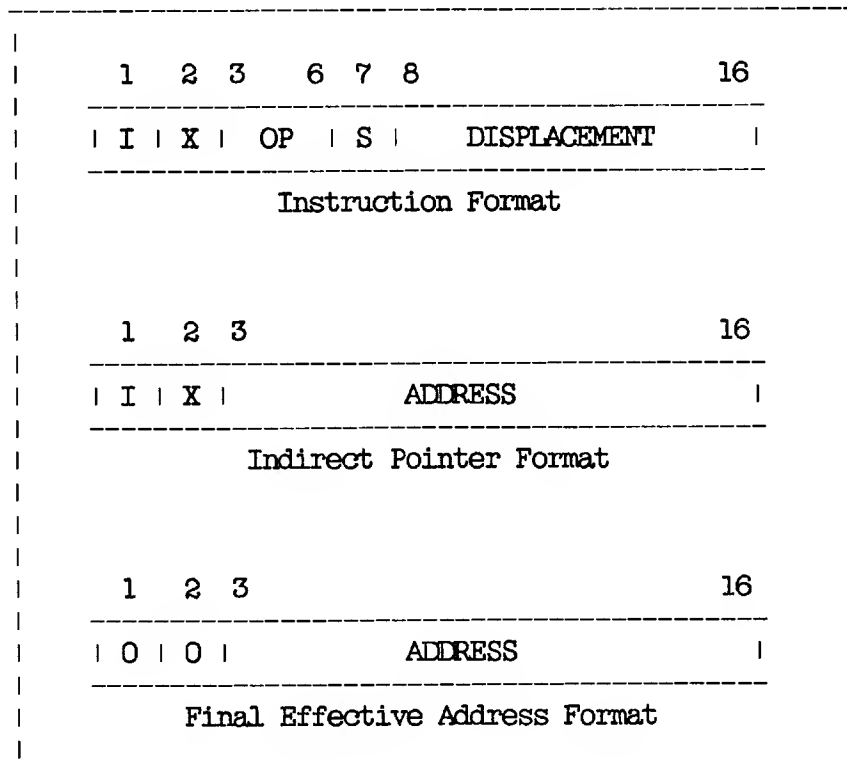
The processor performs one level of indexing and multiple levels of indirection.

O/D indicates that the displacement is within Sector 0.

The instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY do not do indexing. The processor treats the X bit as a 0 to determine what addressing mode to use. For example, if one of these instructions specifies I, X, S, and CB as 0113, the processor interprets it as 0013.

16S Mode

Figure 3-9 and Table 3-9 display and explain 16S mode instructions.



16S Mode Formats

Figure 3-9

Note to Figure 3-9

The final form of effective addresses in S mode are only 14 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide.

Table 3-9
16S Mode Summary

I	X	S	Disp	Instruction Type	Example	EA Form
0	0	0	0 to '777	Direct	LDA ADR	O/D
0	0	1	0 to '777	Direct	LDA ADR	C/D
0	1	0	0 to '777	Indexed	LDA ADR,1	O/D+X
0	1	1	0 to '777	Indexed	LDA ADR,1	C/D+X
1	0	0	0 to '777	Indirect	LDA ADR,*	I(O/D)
1	0	1	0 to '777	Indirect	LDA ADR,*	I(C/D)
1	1	0	0 to '777	Indirect preindexed	LDA ADR,1*	I(D+X)
1	1	1	0 to '777	Indirect preindexed	LDA ADR,1*	I(D+X)

Notes to Table 3-9

The processor performs indexing before resolving each level of indirection.

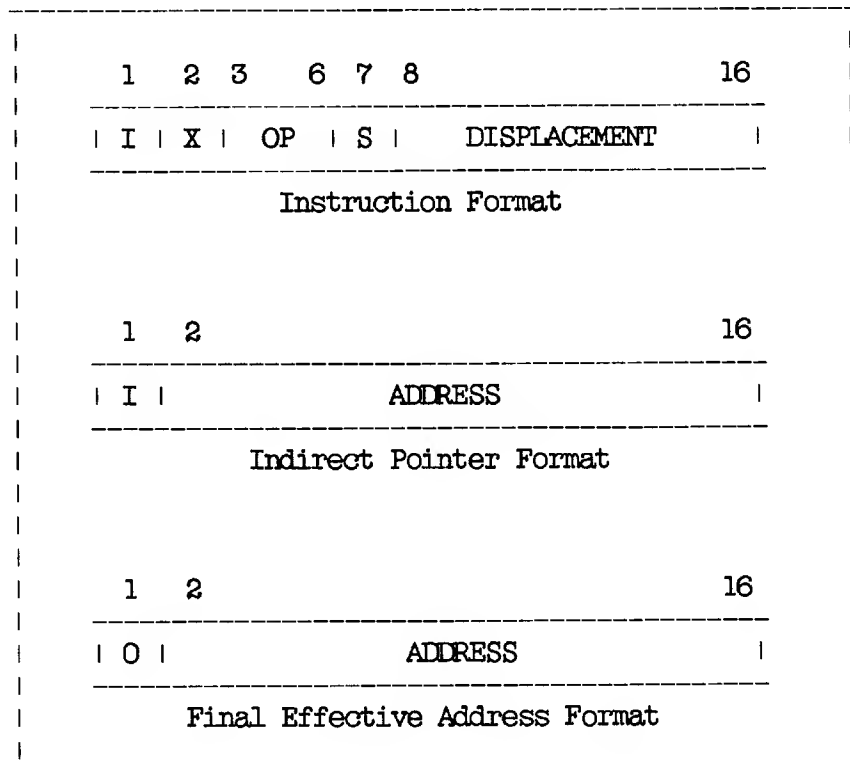
This mode allows multiple levels of both indexing and indirection.

The instructions, LDX and STX, cannot do indexing. The effective address is formed as if bit 2 = 0.

O/D indicates that the displacement is within Sector 0; C/D, within the current sector.

32S Mode

Figure 3-10 and Table 3-10 display and explain 32S mode instructions.



32S Mode Formats

Figure 3-10

Note to Figure 3-10

The final form of effective addresses in S mode are only 15 bits wide. Special hardware exists to truncate the effective address to this length. The program counter, however, is a full 16 bits wide.

Table 3-10
32S Mode Summary

I	X	S	Disp	Instruction Type	Example	EA Form
0	0	0	0 to '777	Direct	LDA ADR	O/D
0	0	1	0 to '777	Direct	LDA ADR	C/D
0	1	0	0 to '777	Indexed	LDA ADR,1	O/D+X
0	1	1	0 to '777	Indexed	LDA ADR,1	C/D+X
1	0	0	0 to '777	Indirect	LDA ADR,*	I(O/D)
1	0	1	0 to '777	Indirect	LDA ADR,*	I(C/D)
1	1	0	0 to '77	Indirect preindexed	LDA ADR,1*	I(D+X)
1	1	0	'100 to '777	Indirect postindexed	LDA ADR,*1	I(D)+X
1	1	1	0 to '777	Indirect postindexed	LDA ADR,*1	I(D)+X

Notes to Table 3-10

The processor performs indexing before resolving each level of indirection.

This mode allows one level of indexing, and multiple levels of indirection.

The instructions, LDX and STX, cannot do indexing. The effective address is formed as if bit 2 = 0.

ADDRESS TRAPS

Several of the summaries in the last section specify special cases of EA formation when the address is within a particular range. This range of addresses corresponds to registers within the current user register set in the register file. (See Chapter 9.) In segmented mode, this range is '0 to '7; in nonsegmented mode, '0 to '37. This range of addresses for segmented and nonsegmented modes is referred to as the ATR, or address trap range, throughout this section.

The registers within the user register set contain information, such as general, base, floating-point, and index registers, and system status and control information. Each time any part of the EA formation generates an address within the ATR, an address trap aborts any read or write to a memory location and instead references the specific register.

Table 3-11 summarizes when address traps occur for all modes of addressing and instruction types.

Table 3-11
Address Trap Information

Mode	Inst Type	Action
16S 32S 32R 64R	Memory reference	Address trap occurs if the EA falls within the ATR (address trap range). The instruction format or length has no bearing.
	Generic	Address traps never occur.
	Generic AP	Address traps do not occur when the processor is fetching the address pointer.
64V	32-bit memory reference	Address traps never occur.
	Short format	See Table 3-12.
	16-bit indirect	Address traps occur if the EA falls within the ATR.
	32-bit indirect	Address traps never occur.
32I	All types	Address traps never occur.

When bits 17 to 32 of the program counter contain a value within the ATR and the processor is reading an instruction, an address trap always occurs. The only exception to this is if the machine is operating in 32I mode.

When the processor executes short format instructions in 64V mode, address traps can occur during operand fetches or indirect fetches. Table 3-12 lists the conditions that must be present for an address trap to occur.

Table 3-12

Address Trap Action for Short Format
Instructions, 64V Mode

I	X	S	Disp	Action
0	0	0	0 to '7	Takes address trap.
0	0	0	'10 to '37	Takes address trap only if segmentation is off.
0	0	0	'40 to '377	Cannot take address trap.
0	0	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.
0	1	0	0 to ATR	Takes address trap if D+X is within the ATR. If D+X is outside the ATR, the EA is SB (seg #) D+X (for the 750, 850, and 2350 to 9955 II; or SB (seg #) D+X+SB (offset #) (for all other machines).
			From ATR to '377	Cannot take address trap; EA is SB+D+X (for 750, 850, and 2350 to 9955 II).
				All other machines take address trap if D+X is within the ATR.
			'400 to '777	Cannot take address trap.
0	1	1	-'340 to +'377	Takes address trap if EA (P+D+X) is within the ATR.
1	0	0	0 to '777	Takes address trap if D is within the ATR.*
1	0	1	-'340 to +'377	Takes address trap if EA ((P+D)) is within the ATR.*
1	1	0	0 to '777	Takes address trap if D<'100 and D+X is within the ATR.*
1	1	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.*

Note to Table 3-12

- * The indirect address also takes an address trap if EA is within the ATR.

If an instruction specifies a write operation that could potentially cause an address trap, the instruction loads the data to be written into a temporary register. If a trap occurs, the routine aborts the write to memory. It loads the specified register file location with the contents of the temporary register.

If the instruction specifies a read operation that causes an address trap, the trap routine aborts the memory read and fetches the contents of a register file location. The trap routine loads the cache from the register file data and allows the processor one cache access before invalidating the cache location.

Table 3-13 shows the address trap locations and the registers to which they correspond. For more information on the register file, see Chapter 9.

Table 3-13
Address Trap/Register File Correspondence

AT	S and R Modes	V Mode
'0	X	X
'1	A	A, LH
'2	B	LL
'3	S	Y
'4	FAC bits 1 to 16	FAC bits 1 to 16
'5	FAC bits 17 to 32	FAC bits 17 to 32
'6	FAC exponent	FAC exponent
'7	PC, LSBs	PC, LSBs
'10*	DTAR3H	DTAR3H
'11*	FOCODEH	FOCODEH
'12*	FADDRL	FADDRL
'13*		
'14*		SBH
'15*		SBL
'16*		LBH
'17*		LBL
'20*	DMA cell '20H	DMA cell '20H
'21*	DMA cell '20L	DMA cell '20L
'22*	DMA cell '22H	DMA cell '22H
'23*	DMA cell '22L	DMA cell '22L
'24*	DMA cell '24H	DMA cell '24H
'25*	DMA cell '24L	DMA cell '24L
'26*	DMA cell '26H	DMA cell '26H
'27*	DMA cell '26L	DMA cell '26L
'30*	DMA cell '30H	DMA cell '30H
'31*	DMA cell '30L	DMA cell '30L
'32*	DMA cell '32H	DMA cell '32H
'33*	DMA cell '32L	DMA cell '32L
'34*	DMA cell '34H	DMA cell '34H
'35*	DMA cell '34L	DMA cell '34L
'36*	DMA cell '36H	DMA cell '36H
'37*	DMA cell '36L	DMA cell '36L

Note to Table 3-13

- * These correspond to user register file locations only in nonsegmented mode.

SUMMARY

The fields of a memory reference instruction specify information used to form an effective address. These fields specify which information is to be used in the formation, how the formation is to be done, and -- in conjunction with the rest of the program -- the addressing mode under which the address is to be formed. Depending on the segmentation mode and the EA formation, addresses can reference registers within the current user register file as well as memory locations.

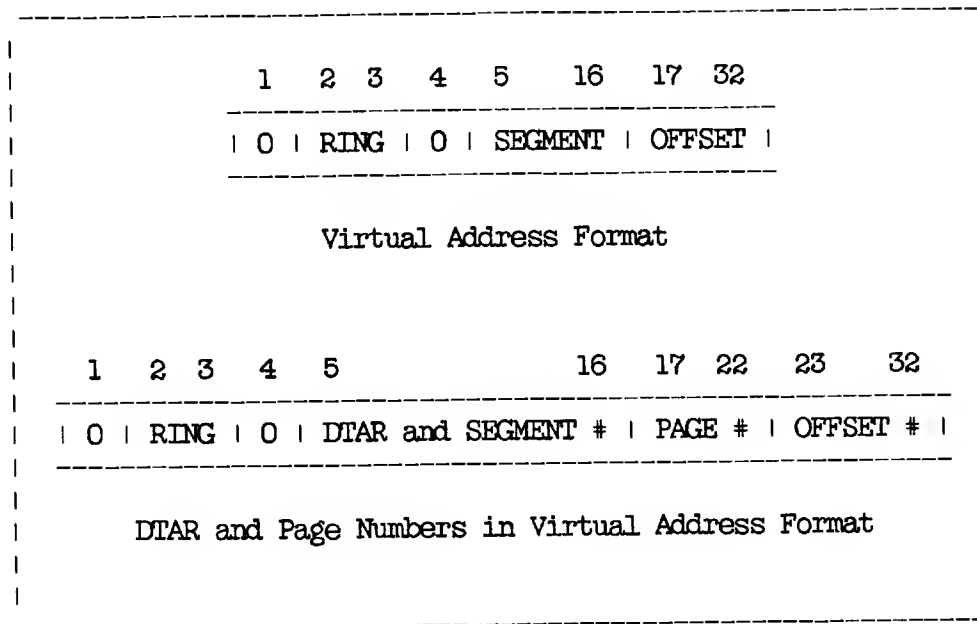
4

Memory Management

The last chapter showed how the 50 Series systems use information contained in an instruction to form a virtual address. This address specifies a location in the virtual address space, which may or may not correspond to a location currently loaded in physical memory. This means that the processor must find some way to convert the virtual address into something that can address a physical memory location, and must then search physical memory for that location. This chapter describes how the processor uses a virtual address to address memory, and describes the data structures (registers and tables) that facilitate the reference.

THE VIRTUAL ADDRESS

A virtual address is a reflection of the segmented virtual address space the user sees. A physical address, similarly, must reflect the pages that make up physical memory. How does the processor make the transition from a segment-oriented address to a page-oriented one? The virtual address (diagrammed in Figure 4-1) is the starting point. (As this figure shows, the page number and DTAR are generally transparent to the user. They are seen only by the mapping hardware.)



Virtual Address Format as Seen by
the Mapping Hardware

Figure 4-1

The steps the processor takes to convert this virtual address into a physical address are:

1. Check the STLB and the cache. If both of these contain the correct information, the reference can be completed. If the STLB contains the correct information but the cache does not, read the information from memory into cache and complete the reference. If the STLB does not contain the correct information, go on to the next step.
2. Translate the virtual address into a physical address. During the translation, identify if the virtual page containing the information is currently loaded into main memory. If it is, load the physical page address (the result of the translation) into the STLB and retry the access. If main memory does not contain the page, go on to the next step.
3. Find the correct virtual page on disk and move it into main memory. After the virtual page is loaded into a physical page, the reference is retried.

The first task is completely performed in hardware; the second, by a microcode routine. A software page fault handler performs all aspects of paging.

MEMORY MANAGEMENT DATA STRUCTURES

All three of the steps in the memory reference operation use several data structures to maintain needed information:

- Segmentation table lookaside buffer (STLB)
- Cache
- Descriptor table address registers (DTARs)
- Segment descriptor tables (SDTs)
- Page map tables (PMTs) for the 2755, 6350, and 9750 to 9955 II
- Hardware page map tables (HMAPs) for all other processors

Table 4-1 shows the steps in which each structure is used.

Table 4-1
Use of Memory Management Data Structures

Structure	When Used
STLB	STLB/cache access, address translation
Cache	STLB/cache access, address translation
DTARs	STLB/cache access, address translation
SDTs	Address translation
PMTs	Address translation, paging (2755, 6350, and 9750 to 9955 II)
HMAPs	Address translation, paging (all other processors)

The STLB

To speed up the virtual to physical address process, the system uses the STLB to store the result of a translation in an STLB entry so that it will have it for reference the next time that the user specifies the same location. Since the STLB has a much faster access time than physical memory does, referencing the STLB saves translation time as well as access time.

The number of entries in the STL_B varies according to processor model as shown in Table 4-2. (Appendix B describes the STL_B of the earlier processors listed on page 1-1.)

Table 4-2
Number of STL_B Entries

Number of STL _B Sets	Entries Per Set	Total Entries	Processors
1	128	128	9750 to 9950
1	512*	512*	2350 to 2755, 9650, 9655, 9955, and 9955 II
2	512	1024	6350

* The 9955 and 9955 II reserve 384 additional entries for segments 0 to 7. Thus, references to these segments are always resident in these processors.

Each STL_B entry specifies one virtual address and one physical page address. Since each entry specifies a physical page address, each STL_B entry is valid for a 2-Kbyte block (one physical page) of physical memory locations. Figure 4-2 and Table 4-3 show the format and content of each STL_B entry.

1	2	3	4	6	7	9	10	19	20	31	32	47		

V M S	RING 1				RING 3			PROC ID			SEG		PHYS ADR	

6350 STLB Entry Format

1	2	3	4	6	7	9	10	21	22	33	34	47		

V M S	RING 1				RING 3			PROC ID			SEG		PHYS ADR	

9955 II STLB Entry Format

1	2	3	4	6	7	9	10	21	22	33	34	46		

V M S	RING 1				RING 3			PROC ID			SEG		PHYS ADR	

9750 to 9955 STLB Entry Format

1	2	3	4	6	7	9	10	19	20	28	29	41		

V M S	RING 1				RING 3			PROC ID			SEG		PHYS ADR	

2755 STLB Entry Format

1	2	3	4	6	7	9	10	19	20	28	29	40		

V M S	RING 1				RING 3			PROC ID			SEG		PHYS ADR	

2350 to 2655, 9650, and 9655 STLB Entry Format

Figure 4-2
STLB Entry Format

Table 4-3
STLB Entry Contents

No. Bits	Mnem	Description
1	V	Valid bit. Indicates if the STLB contains valid data.
1	M	Modified bit. Specifies if the physical page has been modified since its contents were loaded from disk. (0 means modified; 1 means not modified.)
1	S	Shared bit. Inhibits cache.
3	RING 1	The Ring 1 access rights that are to govern the reference.
3	RING 3	The Ring 3 access rights that are to govern the reference.
10 or 12	PROC ID	The process ID of the process making the memory reference. In process exchange, these are the first bits of the offset in segment OWNERH where the process resides.
12 or 9*	SEG	The segment number from the virtual address.
12 to 16**	PHYS ADR	The physical page address (from translation).

* Bits 20 to 28 for the 2350 to 2755, 9650, and 9655 (the upper 9 bits of the segment number).

** Bits 32 to 47 for the 6350.
 Bits 34 to 47 for the 9955 II.
 Bits 34 to 46 for the 9750 to 9955.
 Bits 29 to 41 for the 2755.
 Bits 29 to 40 for the 2350 to 2655, 9650, and 9655.

To access an entry in the STLB, the processor uses a hashing algorithm. The precise algorithm varies according to processor as shown in the rest of this section.

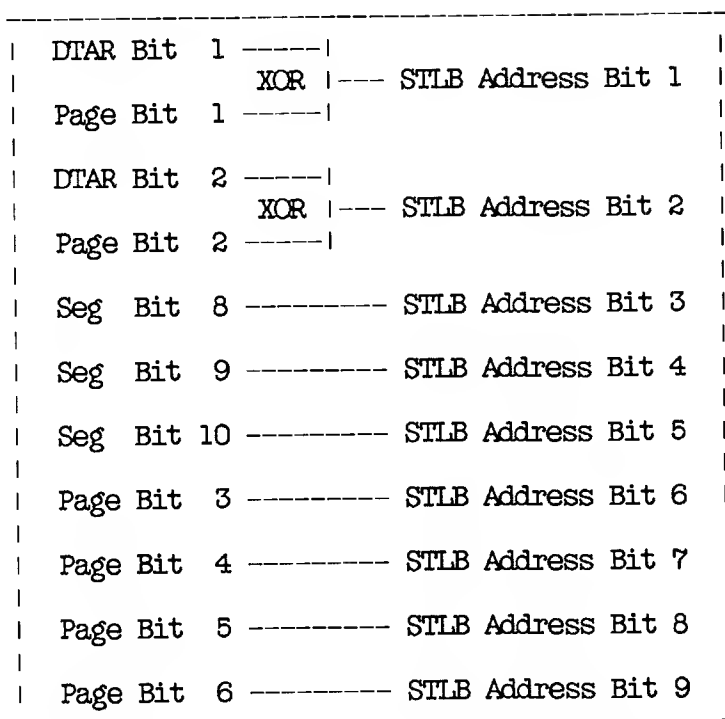
The 6350, 9955, and 9955 II use eleven bits from the virtual address in the hashing algorithm, as shown in Table 4-4. This table also identifies the names that will be used for these bits in the explanation of the algorithm.

Table 4-4
Bits Used in the Hashing Algorithm
For the 6350, 9955, and 9955 II

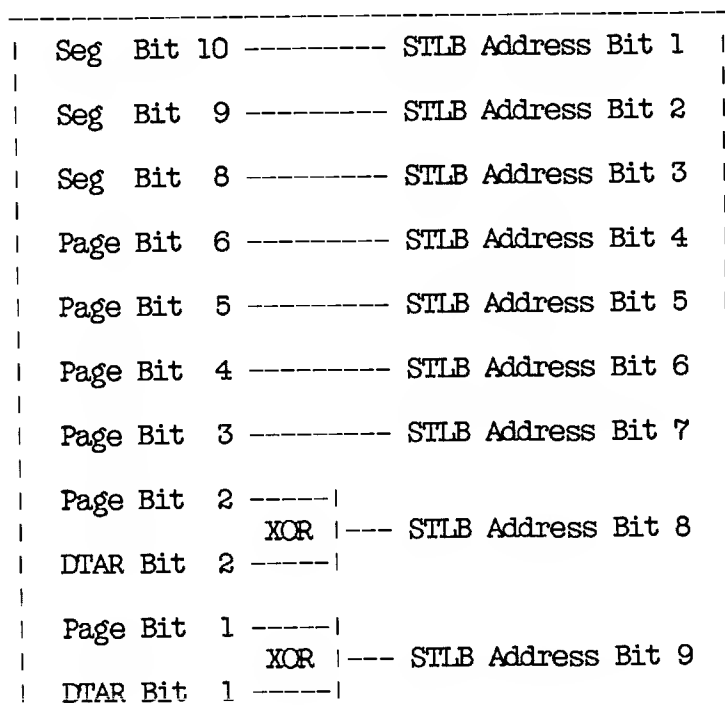
Bits	Name
Bits 5 and 6 of the virtual address. This specifies one of the four DTARS.	DTAR Bit 1 and DTAR Bit 2
Bits 14 to 16 of the virtual address. These are the three least significant bits of the segment field.	Seg Bit 8 to Seg Bit 10
Bits 17 to 22 of the virtual address. These are all of the bits in the page field.	Page Bit 1 to Page Bit 6

From Table 4-4, the hashing algorithm exclusively ORs pairs of bits to form a 9-bit address into the STLB. Figure 4-3 shows how these bits are used to form the STLB entry address for the 6350; Figure 4-4 provides this information for the 9955 and 9955 II.

The 6350, however, has a two-set associative STLB as described in Chapter 1. When the processor has formed the STLB entry address, this address is at the same offset in both parts of the STLB. Therefore, for the 6350, one STLB entry address is used to access two STLB entries. This method minimizes two-way thrashing, which is described in further detail in the section "Accessing the STLB and Cache".



Hashing Algorithm for the STLB of the 6350
Figure 4-3



Hashing Algorithm for the 9955 and 9955 II STLB
Figure 4-4

For the 9750 to 9950, ten bits from the virtual address are used in the hashing algorithm as shown in Table 4-5. This table also contains the names used for these bits in the illustration of the algorithm. From this table, the hashing algorithm exclusively ORs pairs of the bits to form a 7-bit address into the STLB as shown in Figure 4-5.

Table 4-5
Bits Used in the Hashing Algorithm for the 9750 to 9950

Bits	Name
Bits 5 and 6 of the virtual address. These specify one of the four DTARs.	DTAR Bit 1 and DTAR Bit 2
Bits 14 and 15 of the virtual address. These are two of the three least sig- nificant bits of the segment field.	Seg Bit 8 and Seg Bit 9
Bits 17 to 22 of the virtual address. These are all of the bits in the page field.	Page Bit 1 to Page Bit 6

Page Bit 1	-----	
	XOR	-----
DTAR Bit 1	-----	STLB Address Bit 1
Page Bit 2	-----	
	XOR	-----
DTAR Bit 2	-----	STLB Address Bit 2
Page Bit 3	-----	STLB Address Bit 3
Seg Bit 9	-----	STLB Address Bit 4
Page Bit 4	-----	
	XOR	-----
Seg Bit 8	-----	STLB Address Bit 5
Page Bit 5	-----	STLB Address Bit 6
Page Bit 6	-----	STLB Address Bit 7

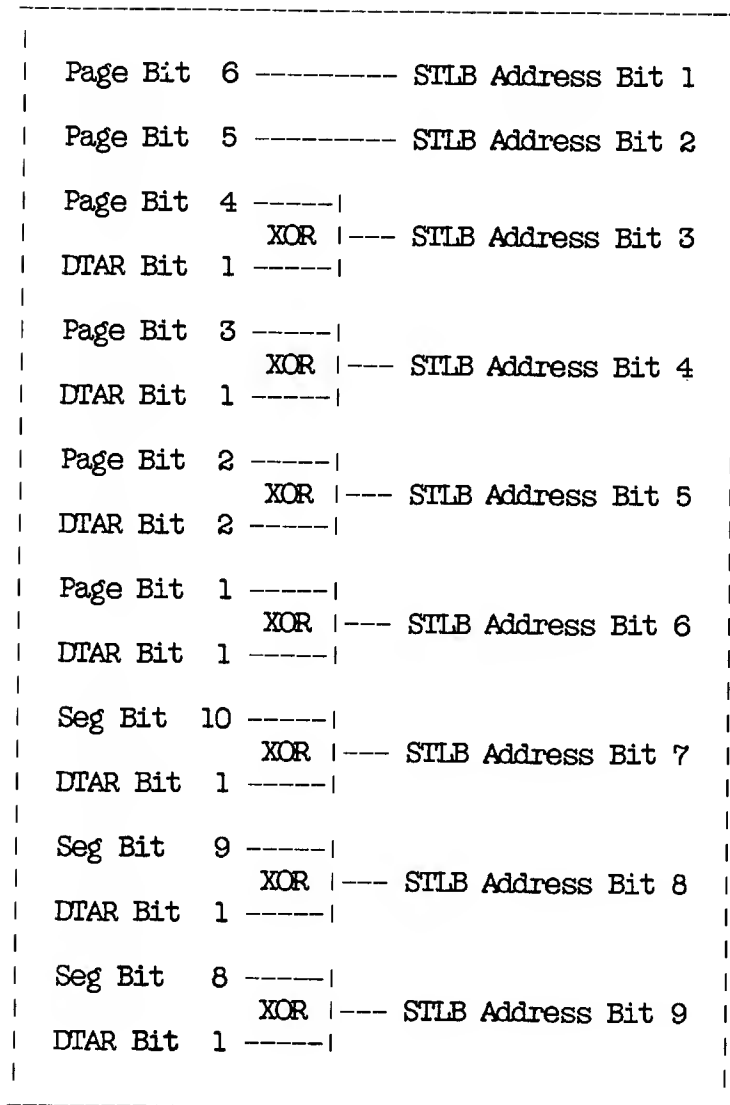
Hashing Algorithm for the 9750 to 9950
Figure 4-5

The 2755 uses eleven bits from the virtual address in the hashing algorithm, as shown in Table 4-6. This table also identifies the names that will be used for these bits in the explanation of the algorithm.

Table 4-6
Bits Used in the Hashing Algorithm for the 2755

Bits	Name
Bits 5 and 6 of the virtual address. This specifies one of the four DTARS.	DTAR Bit 1 and DTAR Bit 2
Bits 14 to 16 of the virtual address. These are the three least significant bits of the segment field.	Seg Bit 8 to Seg Bit 10
Bits 17 to 22 of the virtual address. These are all of the bits in the page field.	Page Bit 1 to Page Bit 6

From Table 4-6, the hashing algorithm exclusive ORs pairs of these bits to form a 9-bit address into the STLB as shown in Figure 4-6.



Hashing Algorithm for the STLB of the 2755
Figure 4-6

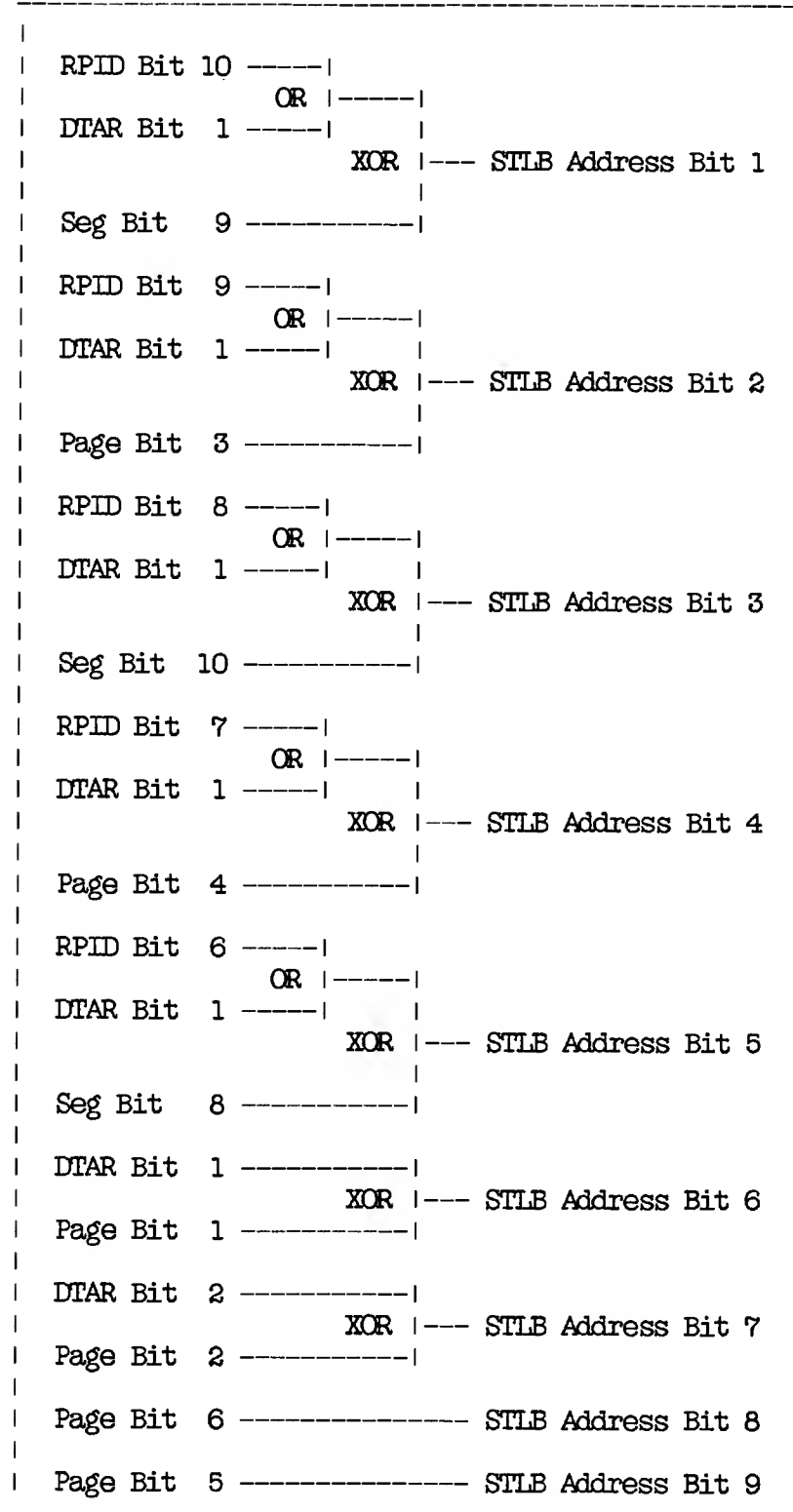
The processors 2350 to 2655, 9650, and 9655 use eleven bits from the virtual address and five bits from the process ID in the hashing algorithm, as shown in Table 4-7. This table also identifies the names that will be used for these bits in the explanation of the algorithm.

Table 4-7

Bits Used in the Hashing Algorithm
For the 2350 to 2655, 9650, and 9655

Bits	Name
Bits 6 to 10 of the process ID. These are the five least significant bits of the process ID.	RPID Bits 6 to 10
Bits 5 and 6 of the virtual address. This specifies one of the four DTARS.	DTAR Bit 1 and DTAR Bit 2
Bits 14 to 16 of the virtual address. These are the three least significant bits of the segment field.	Seg Bit 8 to Seg Bit 10
Bits 17 to 22 of the virtual address. These are all of the bits in the page field.	Page Bit 1 to Page Bit 6

From Table 4-7, the hashing algorithm ORs and exclusive ORs pairs of these bits to form a 9-bit address into the STLB as shown in Figure 4-7.



Hashing Algorithm for the STLB
Of the 2350 to 2655, 9650, and 9655

Figure 4-7

Cache

Like the STLB, the cache specifies the page number of the desired physical location. In addition, it contains the contents of that physical location. Figure 4-8 describes the format of each cache entry. The 6350 has a two-set associative cache, as described later in the section Accessing the STLB and Cache. (See Appendix B for cache entry formats for the earlier processors listed on page 1-1).

1	1	16	1	32
V	PHYSICAL PAGE NUMBER		DATA	

6350 Cache Entry Format

1	1	14	1	32
V	PHYSICAL PAGE NUMBER		DATA	

9955 II Cache Entry Format

1	1	13	1	32
V	PHYSICAL PAGE NUMBER		DATA	

2755 and 9750 to 9955 Cache Entry Format

1	1	12	1	32
V	PHYSICAL PAGE NUMBER		DATA	

2350 to 2655, 9650, and 9655 Cache Entry Format

Number of Bits	Mnemonic	Description
1	Valid	The cache holds valid data when this bit contains 1.
12 to 16	Physical Page Number	Specifies the number of the physical page that contains the specified location. This is the <u>cache index</u> .
32	Data	Contains a copy of the contents of two consecutive locations in physical memory.

Cache Entry Format
Figure 4-8

DTARs

As described in Chapter 2, the 50 Series virtual address space is divided into four groups of 1024 segments each. Each group is referenced through a descriptor table address register (DTAR) associated with it. The public (shared) segments are referenced through DTAR0 and DTAR1; the private (unshared) segments are referenced through DTAR2 and DTAR3. Figure 4-9 shows the format of the DTARs.

1	10 11	16 17	18	32
SIZE	A	-	B	

Bits	Mnem	Description
1 to 10	SIZE	Specifies 1024 minus the size of the segment descriptor table.
11 to 16	A	Bits 1 to 6 of the segment descriptor table physical address.
17	---	Must have the same value as bit 18.
18 to 32	B	Bits 7 to 21 of the segment descriptor table physical address. (Bit 22 of the SDT physical address is 0.)

DTAR Format
Figure 4-9

Segment Descriptor Tables

Each of the four DTARs described above points to a segment descriptor table (SDT). These SDTs contain from 1 to 1024 32-bit entries called segment descriptor words (SDWs). Each SDW describes one segment. The table must begin on an even 16-bit boundary, and must not cross a segment boundary. It must also be located in the first 8 Mbytes of physical memory, since the DTAR can specify only a 22-bit address. The format of the SDWs is shown in Figure 4-10.

1	16	17	18	20	21	23	24	26	27	32

	PHYSICAL ADDRESS		F		A1		---		A3	PHYSICAL ADDRESS

Bits	Mnem	Description
1 to 16	PHYSICAL ADDRESS	Bits 7 to 22 of the physical starting address of a PMT or HMAP. Bits 17 to 22 of this physical starting address must be 0.
17	F	Fault bit.
18 to 20	A1	Specifies the access rights for Ring 1: 000 = no access 001 = gate 010 = read access 011 = read, write access 100 = reserved 101 = reserved 110 = read, execute access 111 = read, write, execute access
21 to 23	---	Reserved.
24 to 26	A3	Specifies the access rights for Ring 3. See bits 18 to 20 for a list of the available access codes.
27 to 32	PHYSICAL ADDRESS	Bits 1 to 6 of the physical starting address of a PMT or HMAP.

Segment Descriptor Word Format
Figure 4-10

Page Map Tables (2755, 6350, and 9750 to 9955 II)

Bits 1 to 16 and bits 27 to 32 of each SDW contain the starting address of a page map table (PMT). These tables contain 64 32-bit entries, each of which contains information about one page. A page map table cannot cross a '200000 (65,536) boundary. Figure 4-11 shows the format of each page map table entry.

1	2	3	4	5	16	17	32
R	U	M	S	SOFTWARE	PAGE ADDRESS*		

Bits	Mnem	Name	Description
1	R	Resident Bit	Indicates if the page resides in physical memory. 1 indicates residency.
2	U	Used bit	Hardware sets U to 1 when a page is used.
3	M	Modified Bit	Hardware resets M to 0 when a page is modified.
4	S	Shared Bit	Inhibits use of cache.
5 to 16	SOFTWARE	Software	Reserved for software use.
17 to 32*	PAGE ADDRESS	Page Address	Specifies high order bits of a physical page address.

* Bits 17 to 18 must be zero for the 9955 II.
 Bits 17 to 19 must be zero for the 2755 and 9750 to 9955.

PMT Entry Format (2755, 6350, and 9750 to 9955 II)
 Figure 4-11

Hardware Page Map Tables (All Other 50 Series Processors)

Bits 1 to 16 and bits 27 to 32 of each SDW contain the starting address of a hardware page map table (HMAP). Each table contains 64 16-bit entries, each of which contains information about one virtual page. An HMAP cannot cross a '200000 (65,536) boundary. Figure 4-12 shows the format of each HMAP entry. This entry is also valid for the earlier systems listed on page 1-1.

1	2	3	4	5	16
R	U	M	S	PAGE ADDRESS	

Bits	Mnem	Name	Description
1	R	Resident Bit	Indicates if the page resides in physical memory. 1 indicates residency.
2	U	Used Bit	Hardware sets U to 1 when a page is used.
3	M	Modified Bit	Hardware resets M to 0 when a page is modified.
4	S	Shared Bit	Inhibits use of cache.
5 to 16	PAGE ADDRESS	Page Address	Specifies high-order 12 bits of physical page address.

HMAP Entry Format (All Other 50 Series Processors)
Figure 4-12

Additional Data Structures

When a referenced virtual page is not in memory, software uses additional mapping data structures to process the resulting page fault. This activity basically includes locating the referenced page on disk, making room for the page in physical memory if necessary by transferring some pages to disk, loading the page into memory, and updating the PMT/HMAP entry for that page. The hardware updates the STLB entry for that page.

ACCESSING THE STLB AND CACHE

As described in Chapter 1, the STLB and the cache are high-speed buffers. If these buffers contain valid information for the process making a reference to a piece of data, the processor can access them in very little time instead of having to make a long memory access.

The hardware accesses both the STLB and the cache in parallel to speed up the reference. A slightly different set of actions is performed, depending on whether the operation is a read or a write. Refer to Figures 4-13 and 4-14 when reading the text in these sections.

Read Memory Access

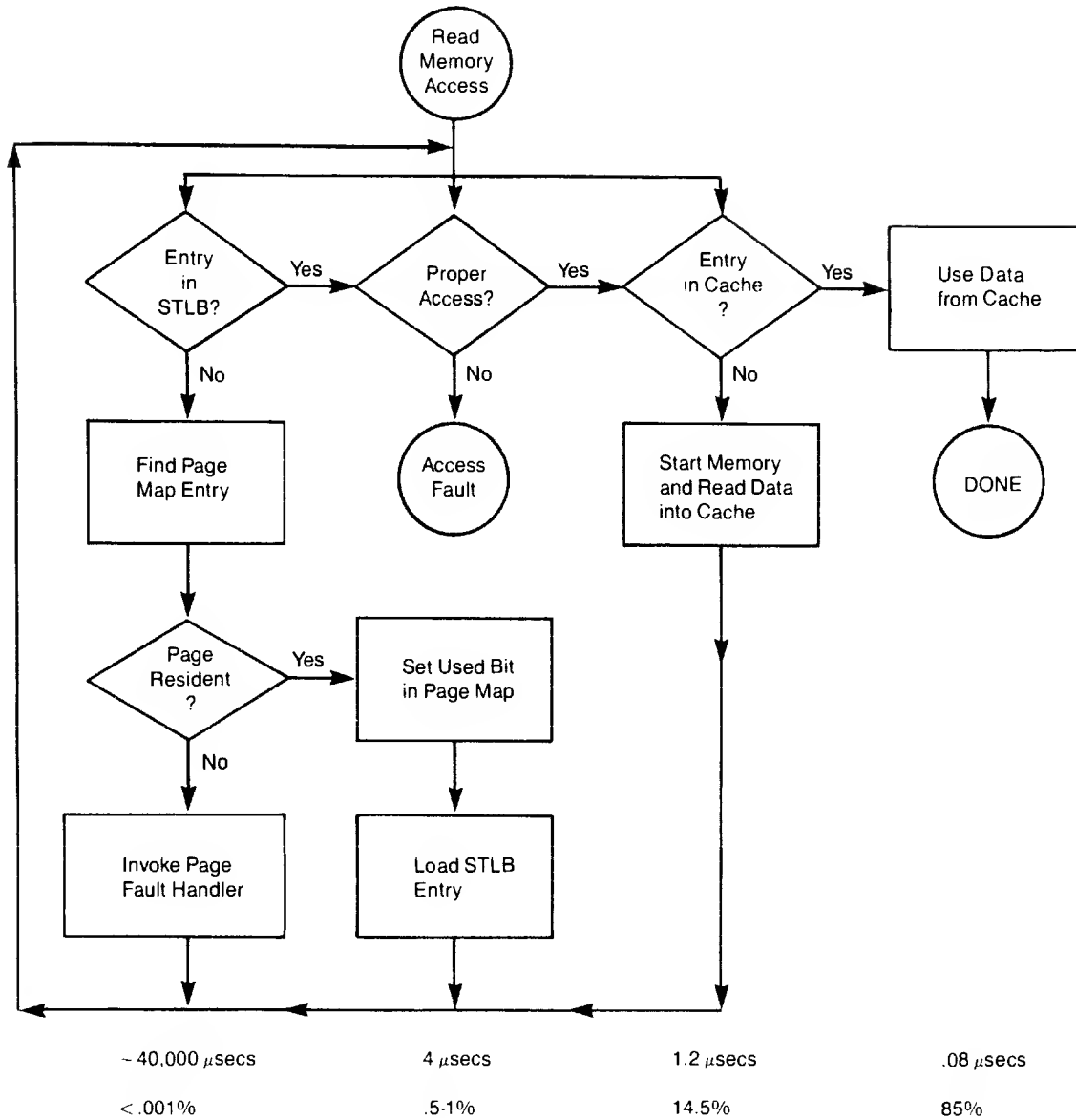
As shown in Figure 4-13, the hardware performs three tasks in parallel: it references the STLB, references the cache, and validates the reference's access rights. The priority among these three tasks is also illustrated in the figure: the leftmost task (checking the STLB entry) has a higher priority than the access check, and the access check has a higher priority than the cache entry step. This means that if a problem arises in the STLB entry step, that is solved first; then the whole access is retried from the beginning. The text in this section describes the access according to this priority.

Step 1. Accessing an STLB Entry

The hashing algorithm described above uses bits from the virtual address to choose an STLB entry. To make sure that this entry contains valid data, the hardware checks the entry's valid bit. If it contains 1, the entry is valid; 0, invalid. The bits in the virtual address' segment number not used to select the STLB entry are compared to the segment number bits for the STLB entry. The hardware must also check that the process ID in the STLB entry is identical to that of the process making the reference. This is done only if the segment number specified in the virtual address is greater than or equal to '4000 -- that is, if the segment specified is in process private address space. If all of these conditions are met, the STLB entry contains valid data and can be used.

For the 6350 processor, the access to the STLB returns two entries. Each entry's segments bits, etc., that are not used in the STLB access described above are compared to the bits in the virtual address. If either of the two entries matches, the valid data from the matching entry is used.

This is called a two-set associative STLB. Each virtual addressing mapping can be in one of two different STLB locations. Two virtual addresses that map to the same STLB address can be used together without references to the one forcing the mapping for the other to be overwritten. When such a conflict occurs, it's called a collision.



Read Memory Access
Figure 4-13

Many collisions can drastically reduce performance; this reduction is called thrashing. The two sets of STLB entries in the 6350 effectively eliminate two-way thrashing.

If the conditions are not met, the STLB needs to be loaded with the correct data. Therefore, the address translation microcode is invoked. (See Address Translation, below.) Assuming no page faults occur, the new translation is loaded into the STLB entry, and the used bit in that entry is set to 1. The reference is then retried from the beginning.

Step 2. Choosing an Access Field

If the STLB entry contains valid data, the hardware must determine what access rights should govern the reference. This requires two steps: first, isolating the ring number that specifies what access field to use; and second, using the access field contents to determine whether the reference is valid or not. STLB entries for segment 0 have no ring field entry and can be accessed only by Ring 0.

To isolate the ring number, the processor weakens the ring number contained within the program counter by logically ORing it with the ring number contained in the effective address. This screens out all invalid references to lower-numbered rings (inward references), but allows references to higher-numbered rings (outward references) to be made.

This screening process makes sure that the access rights of the referencing procedure are weaker than those of the referenced procedure. If this were not done, then a Ring 3 procedure could call a Ring 0 procedure, which in turn could call several procedures for which the Ring 3 procedure had no access rights. Screening out such references protects the integrity of the entire system.

Once the EA ring number has been weakened, the processor uses the weakened ring number to select an access field. If the ring number is 00, the hardware assumes that the reference has unlimited access and no further access checking is done. If the ring number is 01 or 11, the hardware uses the Ring 1 or Ring 3 access fields, respectively, in the STLB entry as the access field. If the ring number is 10, undefined results occur.

The access fields in the STLB entry specify the operations that references using this entry can legitimately perform. Table 4-8 lists the values these fields can contain and their meanings.

Table 4-8
Access Field Values and Their Meanings

Value	Description
000	No access
001	Gate (See Chapter 8.)
010	Read access
011	Read, write access
100	Reserved
101	Reserved
110	Read, execute access
111	Read, write, execute access

The hardware checks the operation specified in the instruction, making the reference against the selected access field to ensure that the operation is valid. For example, if the instruction specifies a read operation and the selected access field allows reads, then the read operation is valid. If, however, the instruction specifies a write and the access field allows only reads, then the operation is invalid. In the first case, the processor performs the valid operation and program execution continues. In the latter case, an access fault occurs and control transfers to the access fault handler. See Chapter 10 for more information about faults.

A reference must have read access to perform either a write or an execute operation. If an instruction specifies either a write or an execute and the access field does not allow reads, an access fault occurs.

Step 3. Accessing the Cache

If the access check is successful, the hardware references the cache. To do this, the hardware must form an address that references an entry in the cache index, which in turn specifies an entry in the cache data. The way that the cache index address is formed depends on the processor as shown in Table 4-9.

Table 4-9
Virtual Address Bits Used in Forming a Cache Index Address

Processors	Virtual Address Bits Used and Significance
2350 to 2655 and 6350 to 9950	20 to 32: the 3 least significant page bits and the 10-bit offset field
2755, 9955, and 9955 II	18 to 32: the 5 least significant page bits and the 10-bit offset field

The 3 and 5 least significant page bits from the virtual address create a virtually mapped cache. See Mapped I/O and DMA in Chapter 11 for information about how the MBIO bits in the IOTLB reconstruct this virtual mapping.

For the earlier processors listed on page 1-1, see Appendix B for cache access information.

When the hardware has an address, it uses it to select an entry, *j*, in the cache index. Entry *j* contains a physical page address, which the hardware compares to the physical page address specified in the STLB entry. If the page numbers are the same, then the *j*th entry in the cache data area contains the contents of the desired physical location. These contents are used in the specified operation.

For the 6350 processor, the access to the cache returns two entries in the cache index. Each of these two entries contains a physical page address, which the hardware compares to the physical page address specified in the STLB entry. If the physical page address contained in either of the two entries matches that specified in the STLB entry, then the data associated with that entry in the cache data area contains the contents of the desired physical location. These contents are used in the specified operation.

This is called a two-set associative cache. The data associated with each virtual address can be in one of two different cache locations. Two virtual addresses with the same cache index address can be used together without references to the one forcing the data for the other to be overwritten. When such a conflict occurs, it's called a collision. Many collisions can drastically reduce performance; this reduction is called thrashing. The two sets of cache entries in the 6350 effectively eliminate two-way thrashing.

If the page numbers are not the same, the hardware must read the data from the physical location specified in the STLB into the cache. It starts memory, reads the data into the cache, and then retries the access from the beginning.

Step 4. Timing Considerations

Figure 4-13 lists the time taken by each step of the read memory access. These figures are based on a 1-MIPS machine. The figure also notes the percentage of times each step is likely to occur. As shown, the cache and STLB contain the needed information 85% of the time, and so the access requires only 80 nanoseconds. In addition, even though a page fault requires 40,000 microseconds it occurs very rarely (on the order of 10 per second). The other three steps occur the majority of the time, and give the system an average read memory access time of .24 to .26 microseconds.

Write Memory Access

Figure 4-14 describes the general steps that occur in a write memory access. The hardware references the STLB, validates the reference's access rights, and checks the STLB modified bit in parallel. The access validation, however, takes precedence over checking the modified bit, and the STLB entry access takes precedence over the access validation. This means that if problems occur in one of the steps with higher precedence, the problem is corrected and the access is retried from the beginning, even if no problems occur with other steps.

Step 1. Accessing the STLB

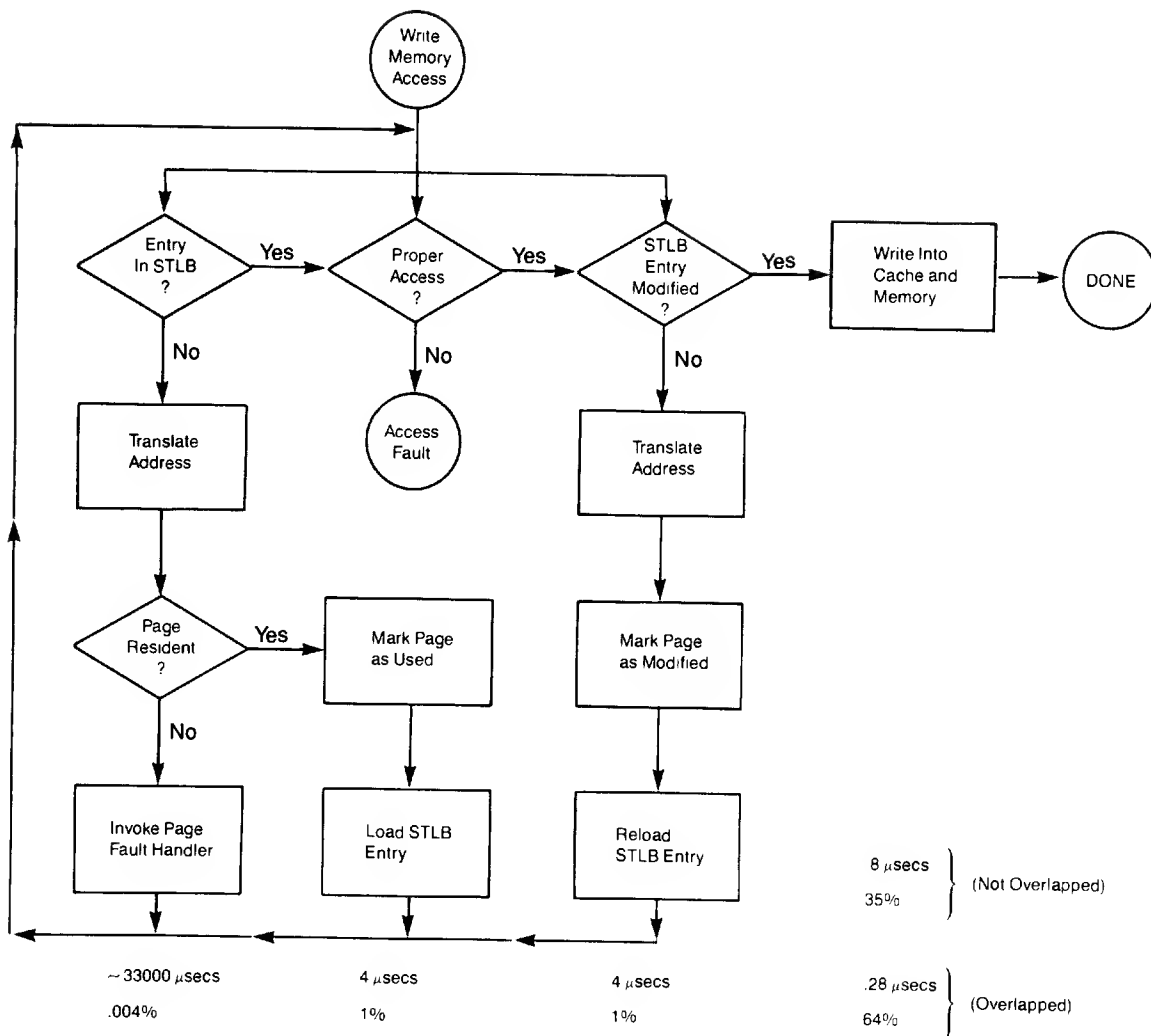
The hardware uses the hashing algorithm described above to select an STLB entry. The entry is validated in the same way as that described in the Read Memory Access section.

Step 2. Checking the Access Rights

This step is identical to that described in the Read Memory Access section above.

Step 3. Checking the STLB Modified Bit

If the STLB entry is valid and if the reference has the proper access, the hardware checks the STLB entry's modified bit. If this bit contains 1, the page is being modified for the first time since this STLB entry was last used. This means that hardware must reset the modified bit in the page map table (PMT or HMAP) and the STLB using the address translation mechanism. Once the new translation is loaded into the STLB entry, the reference is retried from the beginning.



Write Memory Access
Figure 4-14

If the STLB entry's modified bit is 0 (meaning this page has been modified), the hardware forms the address of a cache entry (see Accessing the Cache, above), starts memory, and writes the contents of the referenced location into memory. The data is also written to cache in all cases on some processors. On other processors, the data is written to cache only if there is a cache hit.

Step 4. Timing Considerations

Figure 4-14 lists the time each step of the write memory access takes. These figures are based on a 1-MIPS machine. The figure also notes the percentage of times each step is likely to occur. As shown, the STLB contains the needed information 35% to 64% of the time, depending on whether the accesses are overlapped or not. In the case of overlapped transfers, the system's average write access time varies from one processor to another, but ranges from about 0.22 to 0.28 microseconds; for transfers that are not overlapped, the average time ranges from about 0.32 to 0.8 microseconds.

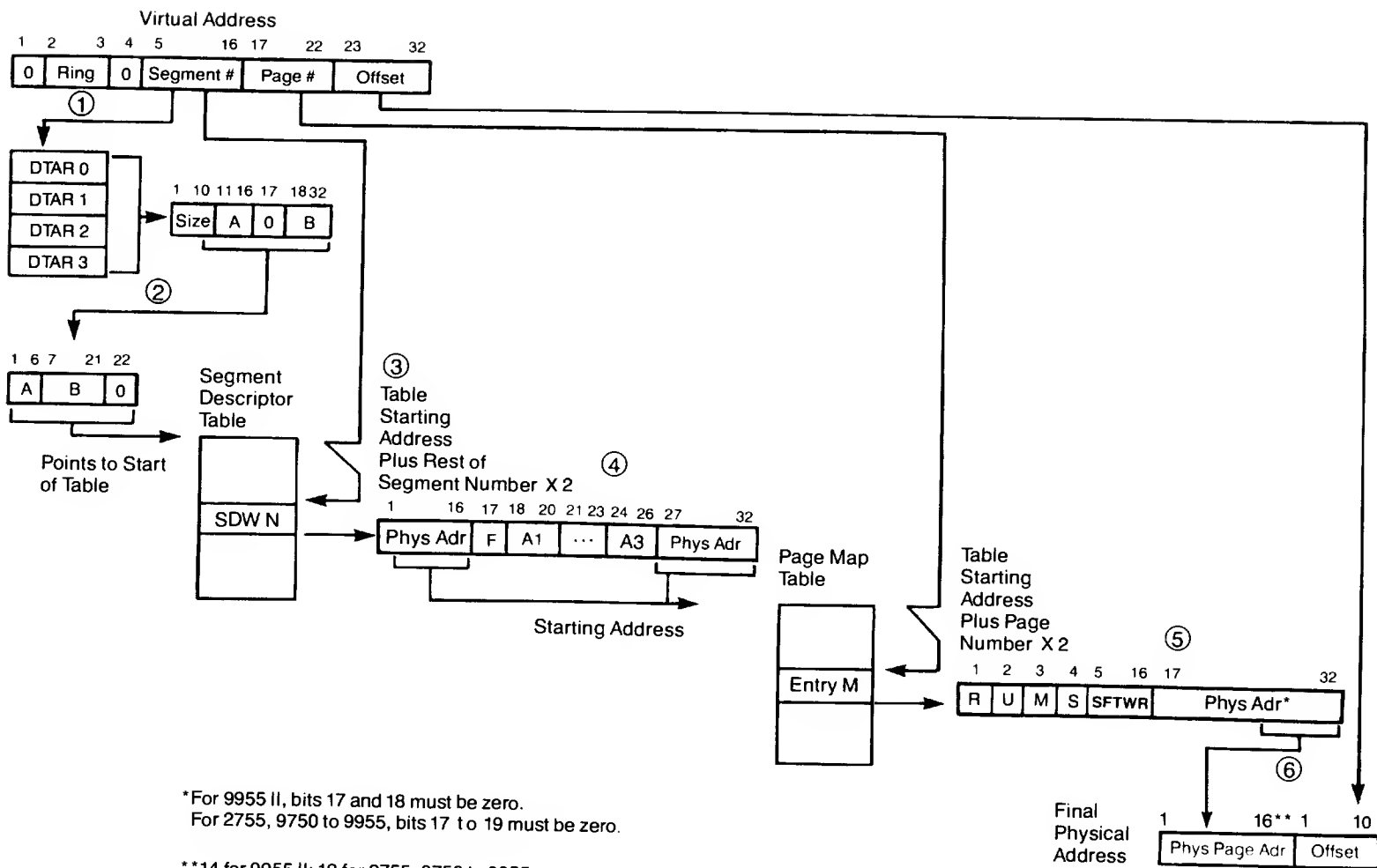
ADDRESS TRANSLATION

When the STLB does not contain information about the virtual-to-physical translation, a microcoded part of PRIMOS (called the address translation mechanism, or ATM) must perform the translation. The DTARs, the segment descriptor tables, and the hardware/page map tables allow the ATM to make the correct reference.

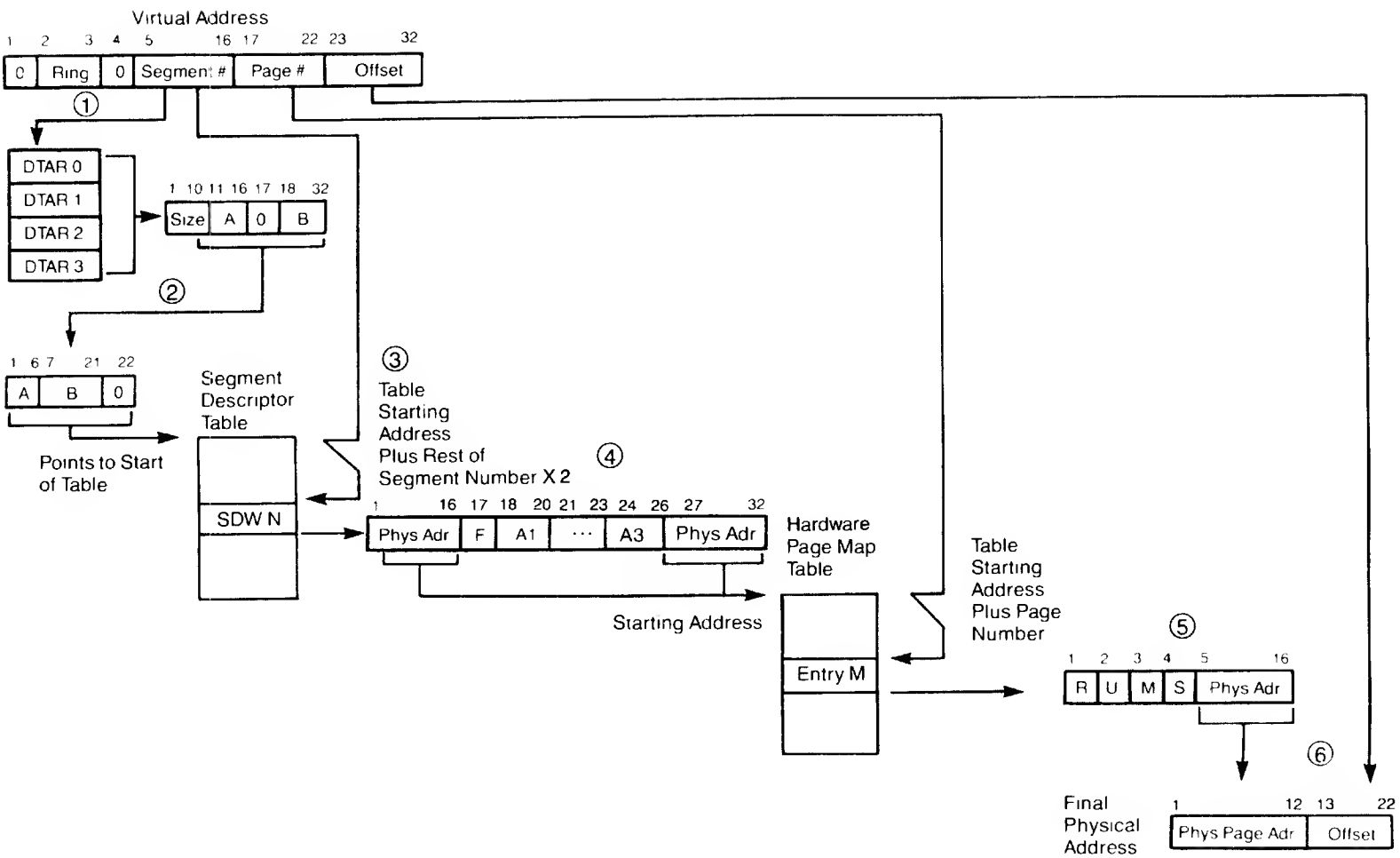
When reading the detailed description of the translation process, refer to Figures 4-15 and 4-16. Figure 4-15 depicts address translation on the 2755, 6350, and 9750 to 9955 II. Figure 4-16 shows address translation on all other processors. The numbers labelling the discussion match the numbers on the diagram.

1. Interpreting the Virtual Address

The virtual address derived from the information contained in an instruction is a 32-bit quantity. When the translation occurs, the virtual address is interpreted as shown in Figure 4-1. Bits 2 to 3 contain protection information and will be described in the next chapter. Bits 5 to 16 contain a segment number; bits 17 to 22, a page number; and bits 23 to 32, an offset. The ATM looks at bits 5 to 6 first, since they specify one of the four DTARs. The ATM references the specified DTAR.



Address Translation on the 2755, 6350, and 9750 to 9955 II
Figure 4-15



Address Translation on Other 50 Series Machines
Figure 4-16

2. Referencing the DTAR

The specified DTAR contains the address of a segment descriptor table, as well as the size of the table. The ATM uses the contents of bits 11 to 32 of the DTAR to form the starting address of the SDT.

3. Validating the Segment Number

After forming the table's starting address, the ATM uses bits 7 to 16 of the virtual address as an offset into the table. It first compares the segment number contained in these bits to bits 1 to 10 of the DTAR to check if the virtual address specifies an invalid segment. If the segment number is greater than the maximum allowable table size, the segment number is invalid and a segment fault occurs (segment number too large). If the segment number is less than or equal to the maximum allowable table size, the segment number is valid and the ATM adds twice the value of virtual address bits 7 to 16 to the starting address of the SDT. The sum specifies an entry, n, in the SDT.

4. Referencing the SDT

Entry n in the SDT contains a segment fault bit, access information (see next chapter), and the address of a Page Map Table (PMT) or a Hardware Page Map Table (HMAP). The ATM checks bit 17, the fault bit, for an invalid segment. If F contains a 1, the segment is invalid or a PMT/HMAP is missing, and a segment fault occurs. If F contains a 0, the segment is valid, and the ATM uses bits 1 to 16 and 27 to 32 of entry n as the starting address of a PMT or an HMAP.

To access the PMT, the ATM adds twice the value of bits 17 to 22 of the virtual address to reference the correct entry, m. To access the HMAP entry m, the ATM adds the contents of bits 17 to 22 of the virtual address to the starting address of the HMAP.

5. Checking Page Status

Bits 1 to 4 of PMT/HMAP entry m contain status information about a page of memory. When the entry is obtained from memory, the ATM examines the used (U) bit. If the content is 1, the page is assumed to be resident (R bit=1). If the U bit content is 0, the resident (R) bit is examined. If R contains 1, the page is resident but unused; the ATM sets the U bit in the PMT/HMAP entry and loads the translation into the STLB. If R contains 0, the page is not resident and a page fault occurs. (Chapter 10 contains more information about faults.) This ordering of the examination of the U and R bits maximizes the speed of the ATM.

Note

The combination of R=0 and U=1 is illegal and will cause undefined results.

6. Forming the Address and Loading the STLB

After determining that no page fault exists, the ATM combines the physical page address in the PMT/HMAP with the 10-bit offset from the virtual address to form a final physical address as shown in Table 4-10.

Table 4-10
Forming the Final Physical Address

Processors	Bits Used by ATM	Final Physical Address Length
6350	PMT bits 17 to 32 and VA bits 23 to 32	26 Bits
9955 II	PMT bits 19 to 32 and VA bits 23 to 32	24 Bits
2755 and 9750 to 9955	PMT bits 20 to 32 and VA bits 23 to 32	23 Bits
All other processors	HMAP bits 5 to 16 and VA bits 23 to 32	22 Bits

The ATM loads this final physical address, plus its associated access information, into the STLB. The translation process for any address has to be done only the first time that any location on the page is referenced, because after that the STLB contains the translated value.

SUMMARY

This chapter described how a 50 Series system uses a virtual address to locate information in physical memory. The cache and STLB provide rapid means of locating commonly referenced information without requiring memory access. When these buffers do not contain the desired information, a combination of processor hardware, firmware, and PRIMOS software can translate the user's virtual address into a physical one through the use of specialized data structures and algorithms. The software page fault handler ensures that information currently on disk is moved in a controlled fashion into main memory when it is needed.

5

Control Information and Restricted Instructions

The previous three chapters have described physical and virtual memory, how they are manipulated, and the data structures used in their manipulation. These data structures, like many parts of PRIMOS, are essential to system operation and so are protected against use by the casual user. However, a set of restricted instructions is available for situations that require manipulation of these and other system structures.

This chapter describes some of these other data structures, especially the keys and modals, and lists the restricted instructions and describes what they do. Restricted instructions can be executed in Ring 0, and many of them perform system functions, such as purging an STLB entry. Others manipulate some of the other system data structures, such as the keys register or the sense switches. For more detailed information about these instructions, refer to the appropriate entries in the Instruction Sets Guide.

OTHER SYSTEM DATA STRUCTURES

There are other data structures the system uses:

- Modals
- Keys
- CBIT, LINK, and condition code bits

Modals

The 16-bit register called the modals contains information about the state of the processor. This register specifies information needed by the hardware and the operating system, such as the type of process control the system uses and which user register set is currently active. (See Chapter 9.) This register is directly accessible only in V and I modes.

Figure 5-1 shows the normal setting of the modals that PRIMOS uses. Figure 5-2 shows the format of the modals. Table 5-1 lists the instructions that modify the modals. Never modify the modals with the STLR instruction; use only the instructions listed in Table 5-1. In addition, never use LPSW to change bits 9 to 11 of the modals. For more information, refer to individual instruction descriptions in the Instruction Sets Guide.

1	8	9	11	12	16

	11000000		CRS		11111

Normal Modals Setting
Figure 5-1

1	2	3	8	9	11	12	13	14	15	16
E	V	000000	CRS	MIO	PXM	S	MCM			

Bits	Mnem	Description
1	E	Enable interrupts: 0 = interrupts disabled 1 = interrupts enabled
2	V	Vectored interrupt mode: 0 = standard interrupt mode 1 = vectored interrupt mode
3 to 8	---	Must be zero.
9 to 11	CRS	Specifies the current register set. Only the PXM can alter these bits. (See Chapter 9.)
12	MIO	Specifies the current mode of I/O: 0 = unmapped mode 1 = mapped mode
13	PXM	Process exchange enable/disable: 0 = process exchange disabled 1 = process exchange enabled
14	S	Specifies the mode of segmentation: 0 = no segmentation 1 = segmentation
15 to 16	MCM	Machine check mode: 00 = no reporting 01 = report only uncorrected memory parity errors 10 = report only unrecovered errors 11 = report all errors See Chapter 10 for more information.

Modals Format
(V and I Modes Only)

Figure 5-2

Table 5-1
Modals Instructions

Mnem	Name	Modes	Description
EMCM	Enter Machine Check Mode	S,R,V,I	Enters machine check mode.
ENB	Enable Interrupts	S,R,V,I	Sets bit 1 of the modals to 1.
ESIM	Enter Standard Interrupt Mode	S,R,V	Resets bit 2 of the modals to 0.
EVIM	Enter Vectored Interrupt Mode	S,R,V	Sets bit 2 of the modals to 1.
INH	Inhibit Interrupts	S,R,V,I	Resets bit 1 of the modals to 0.
LMCM	Leave Machine Check Mode	S,R,V,I	Leaves machine check mode.
LPSW	Load Program Status Word	V,I	Loads the PSW with the contents of a location in memory.
RMC	Reset Machine Check Flag to 0	S,R,V,I	Resets bits 15 to 16 of the modals to 0 and inhibits interrupts for the next instruction.

Keys

The other 16-bit register, the keys, describes the currently running process and the procedure that process is executing. The keys contain status information (such as the mode of addressing currently enabled) and specify fault handling information. Figure 5-3 shows the format of the keys for S mode and R mode. Figure 5-4 shows the format for V and I modes, and Figure 5-5 displays the normal settings for the V and I mode keys. Table 5-2 lists the instructions that modify the keys.

Never modify the keys or modals with the STLR instruction; use only the instructions listed in Table 5-2. In addition, never use LPSW to change bits 15 to 16 of the keys. For more information, refer to individual instruction descriptions in the Instruction Sets Guide.

CONTROL INFORMATION AND RESTRICTED INSTRUCTIONS

1	2	3	4	6	7	8	9	16
CBIT	DBL	--	MODE	FEX	IEX	VISIBLE	SHIFT	COUNT

Bits	Mnem	Description
1	CBIT	Reflects arithmetic conditions of some instructions.
2	DBL	Reflects arithmetic mode: 0 = single precision 1 = double precision
3	---	Reserved for future use.
4 to 6	MODE	Specifies the current mode of addressing: 000 = 16S 001 = 32S 010 = 64R 011 = 32R 100 = 32I 101 = unused 110 = 64V 111 = unused
7	FEX	Floating-point exception enable/disable: 0 = set CBIT to 1 and invoke fault handler on error 1 = set CBIT to 1 only on error
8	IEX	Integer exception enable/disable: 0 = set CBIT to 1 only on error 1 = set CBIT to 1 and invoke fault handler on error
9 to 16	VISIBLE SHIFT COUNT	Bottom half of the floating-point exponent.

Keys Format, S and R Modes
Figure 5-3

1	2	3	4	6	7	8	9	10	11	12	13	14	15	16
CBIT	0	LINK	MODE	FEX	IEX	LT	EQ	DEX	ASCII-8	RND	P850	IN	SD	

Bits	Mnem	Description
1	CBIT	Reflects arithmetic conditions of some instructions.
2	---	Must be zero.
3	LINK	Reflects arithmetic conditions of some instructions.
4 to 6	MODE	Specifies the current mode of addressing: 000 = 16S 001 = 32S 010 = 64R 011 = 32R 100 = 32I 101 = unused 110 = 64V 111 = unused
7	FEX	Floating-point exception enable/disable: 0 = set CBIT to 1 and invoke fault handler on error 1 = set CBIT to 1 only on error
8	IEX	Integer exception enable/disable: 0 = set CBIT to 1 only on error 1 = set CBIT to 1 and invoke fault handler on error
9	LT	Less Than condition code: 1 reflects a less than 0 condition.
10	EQ	Equal To condition code: 1 reflects an equal to 0 condition.
11	DEX	Decimal exception enable/disable: 0 = set CBIT to 1 only on error 1 = set CBIT to 1 and invoke fault handler on error

Keys Format, V and I Modes
Figure 5-4

Bits	Mnem	Description
12	ASCII-8	ASCII character representation: specifies whether set or reset ASCII characters are to be generated. 0 = most significant bit of characters is 1 (set format) 1 = most significant bit of characters is 0 (reset format) Disregarded on the earlier Prime systems (listed on page 1-1)
13	RND	Floating-point round: specifies the form of rounding to use in floating-point operations. 0 = no rounding 1 = rounding Disregarded on the earlier Prime systems (listed on page 1-1)
14	P850	P850 bit: used by the P850 processor. (See Appendix C for P850 information.) This bit may be used for other processor-specific features.
15	IN	In dispatcher: specifies if the current process associated with the register is in the dispatcher. 0 = process is not in the dispatcher 1 = process is in the dispatcher Only the PKM (process exchange mechanism) alters this bit.
16	SD	Save done bit: specifies if PKM has saved values of current register set. 0 = save must be done before this register set can be used 1 = save has been done and this register set is available Only the PKM alters this bit.

Keys Format, V and I Modes
Figure 5-4 (continued)

1	3 4	6 7	16
AOA	110 or 100*	00AA000000	

* Bits 4 to 6 are 110 (V mode) or 100 (I mode).

A - The value can be altered by an instruction.

Normal Keys Setting in V and I Modes
Figure 5-5

Table 5-2
Keys Instructions

Mnem	Name	Modes	Description
DBL	Enter Double Precision Mode	S,R	Sets bit 2 in the keys to 1.
E16S	Enter 16S Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 000.
E32I	Enter 32I Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 100.
E32S	Enter 32S Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 001.
E32R	Enter 32R Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 011.
E64R	Enter 64R Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 010.
E64V	Enter 64V Mode	S,R,V,I	Sets bits 4 to 6 of the keys to 110.
INK	Input Keys	S,R,I	Reads the keys into the specified register.
OTK	Output Keys	S,R,I	Loads the keys with the contents of the specified register.
RCB	Reset CBIT	S,R,V,I	Resets the value of CBIT in the keys to 0.
SCA	Load Shift Count into A	S,R	Loads bits 9 to 16 of the keys into bits 9 to 16 of A.
SCB	Set CBIT	S,R,V,I	Sets the value of CBIT in the keys to 1.
SGL	Enter Single Precision Mode	S,R	Sets bit 2 in the keys to 0.
LPSW	Load PSW	V,I	Loads new data into the keys, modals, and program counter.
TAK	Transfer A to Keys	S,R,V	Transfers the contents of A into the keys.
TKA	Transfer Keys to A	S,R,V	Transfers the contents of the keys into A.

CBIT, LINK, and the Condition Codes

Some of the bits in the keys merit extra discussion. Bit 1, CBIT, and bit 3, LINK, are set by many instructions to indicate conditions under which the instruction completed execution. Several instructions performing arithmetic operations, for example, set CBIT to 1 to indicate that the operation has resulted in an overflow (a result too large to fit in the specified number of bits). Others set LINK to 1 to reflect a carry out condition. Still others set CBIT to indicate a fault condition. The instruction entries in the Instruction Sets Guide state how each instruction affects the values of these bits.

Also note that bits 9 and 10 of the keys contain the condition codes. Many arithmetic, branch, skip, jump, and other instructions set these bits to indicate the result of a test (a result is less than 0, for example), to indicate whether a value is positive or negative, and so on. Other instructions use the condition code values as Boolean values. The instruction entries in the Instruction Sets Guide also describe how an instruction affects the state of these bits.

EQ shows whether or not a 16-bit or 32-bit result is equal to 0. LT contains the extended sign for arithmetic and comparison operations. The extended sign is the sign of the result as if the operation had been done on a machine of infinite precision; thus, LT shows the correct sign of the result despite any overflow. For logic operations, LT reflects the sign of the result. Table 5-3 shows condition code interpretation for comparison, arithmetic, and logic operations.

The state of the CBIT, LINK, and condition codes is recorded in special hardware after each instruction that modifies them. These are referred to as the live keys and are the values tested by instructions. The keys register obtains a copy of the live keys upon updates, but may not reflect the actual state of the CBIT, LINK, or condition codes. The state of these bits should only be tested for with the appropriate instruction.

Table 5-3
Interpretation of Condition Codes

LT, EQ Values	Comparison	Arithmetic	Logic
00	Register > 0 Register > EA Reg 1 > Reg 2	Signed result > 0 Unsigned result <> 0	Result <> 0, High-order bit = 0
01	Register = 0 Register = EA Reg 1 = Reg 2	Result = 0	Result = 0, High-order bit = 0
10	Register < 0 Register < EA Reg 1 < Reg 2	Result < 0	Result <> 0, High-order bit = 1
11	Not working	Happens only when the largest negative number is added to itself. (CBIT is set to 1 as well to show overflow, or is loaded with this state by TAK, LPSW, process exchange, PCL, etc.	Not working

RESTRICTED INSTRUCTIONS

Table 5-4 lists the restricted instructions and briefly describes their actions. Refer to the Instruction Sets Guide for more information about these instructions.

Table 5-4
Restricted Instructions

Mnem	Name	Modes	Description
EIO	Execute I/O	V,I	Executes an effective address as an I/O instruction.
ENB	Enable Interrupts	S,R,V,I	Enables interrupts so that devices can request service.
HLT	Halt	S,R,V,I	Halts the processor.
INA	Input to A	S,R	Loads data from the specified device into A.
INBC	Interrupt Notify	V,I	Notifies during the interrupt code. Uses LIFO queuing. Clears the currently active interrupt.
INBN	Interrupt Notify	V,I	Notifies during the interrupt code. Uses LIFO queuing. Does not clear the currently active interrupt.
INEC	Interrupt Notify	V,I	Notifies during the interrupt code. Uses FIFO queuing. Clears the currently active interrupt.
INEN	Interrupt Notify	V,I	Notifies during the interrupt code. Uses FIFO queuing. Does not clear the currently active interrupt.
INH	Inhibit Interrupts	S,R,V,I	Disables interrupts so that devices cannot request service.
IRTC	Interrupt Return	V,I	Returns control from an interrupt and clears the currently active interrupt.
IRTN	Interrupt Return	V,I	Returns control from an interrupt and does not clear the currently active interrupt.
ITLB	Invalidate STLB Entry	V,I	Invalidates the STLB entry specified by L.
LIOT	Load I/O TLB	V,I	Loads an entry in the IOTLB.

Table 5-4 (continued)
Restricted Instructions

Mnem	Name	Modes	Description
LPID	Load Process ID	V,I	Loads the process ID contained in A into RPID.
LPSW	Load PSW	V,I	Loads new values into the program counter, keys, and modals.
NFYE	Notify End of Queue	V,I	Notifies on the specified semaphore. Uses LIFO queuing. Does not clear the currently active interrupt.
NFYB	Notify Head of Queue	V,I	Notifies on the specified semaphore. Uses FIFO queuing. Does not clear the currently active interrupt.
OCP	Output Control	S,R	Sends a control pulse to a device.
OTA	Output from A	S,R	Transfers data from A to the specified device.
PTLB	Purge TLB	V,I	Purges either an entry or a page in the translation lookaside buffer.
RMC	Clear Machine Check	S,R,V,I	Clears the machine check flag.
RTS	Reset Time Slice	V,I	Resets the value of the interval timer.
SKS	Skip on Satisfied Condition	S,R	When the specified condition is satisfied, the specified device responds ready and SKS skips the next 16 bits.
STPM	Store Processor Model Number	V,I	Stores the CPU model number and microcode revision number into memory.
WAIT	Wait	V,I	Waits until the specified semaphore is notified.

SUMMARY

In this chapter you have read about more of the system registers and data structures that aid in controlling system operation. The next chapter, Datatypes, presents the data representations and formats supported on the 50 Series processors. It also lists the instructions you can use to manipulate the various types of data.

6

Datatypes

The 50 Series systems support several data representations. These representations fall into the major groups:

- Fixed-point data
- Floating-point numbers
- Decimal integers
- Character strings
- Queues

This chapter describes each of these data representations, and the operations and instructions available to manipulate each type.

Throughout the rest of this book, R is used to indicate a 32-bit I mode general register, while r indicates bits 1 to 16 of a 32-bit I mode general register. In addition, A and B represent S and R mode 16-bit registers; L and E represent V mode 32-bit registers.

FIXED-POINT DATA

Fixed-point data can be a logical value, a signed or unsigned integer, or an address. Addresses are treated as unsigned integers.

Logical Values

A logical value is a 16-bit or 32-bit value that is interpreted as a string of bits. Table 6-1 lists the instructions that perform logical operations, such as OR and AND. The 50 Series processors treat each bit in a bit string separately: the value of one bit does not affect the value of another.

There are several instructions available that test logical values and perform an action depending on the result of the test. Chapter 7 discusses these instructions.

Table 6-1
Logic Instructions

Mnem	Name	Modes	Description
ANA	AND to A	S,R,V	Logically ANDs the contents of A and the contents of a memory location.
ANL	AND Long	V	Logically ANDs the contents of L and the contents of a memory location.
CMA	Complement A	S,R,V	Forms the one's complement of the contents of A.
CMH	Complement Halfword	I	Forms the one's complement of the contents of r.
CMR	Complement Fullword	I	Forms the one's complement of the contents of R.
ERA	Exclusive OR to A	S,R,V	Exclusively ORs the contents of A and the contents of a memory location.
ERL	Exclusive OR Long	V	Exclusively ORs the contents of L and the contents of a memory location.
N	AND Fullword	I	Logically ANDs the contents of R and the contents of a memory location.
NH	AND Halfword	I	Logically ANDs the contents of r and the contents of a memory location.
O	OR Fullword	I	Logically ORs the contents of R and the contents of a memory location.
OH	OR Halfword	I	Logically ORs the contents of r and the contents of a memory location.
ORA	Inclusive OR to A	V	Logically ORs the contents of A and the contents of a memory location.
X	Exclusive OR Fullword	I	Exclusively ORs the contents of R and the contents of a memory location.
XH	Exclusive OR Halfword	I	Exclusively ORs the contents of r and the contents of a memory location.

Signed Integers

Depending on the addressing mode, there are a variety of signed integer formats to use. Each is based on a magnitude field that represents a two's complement value. Figure 6-1 shows the formats and data sizes available for each addressing mode.

Size	Modes	Format
16 bits	S,R, V,I	<div> <div>1</div> <div>16</div> <div>MAGNITUDE</div> </div>
32 bits	V,I	<div> <div>1</div> <div>32</div> <div>MAGNITUDE</div> </div>
64 bits	V,I	<div> <div>1</div> <div>64</div> <div>MAGNITUDE</div> </div>
31 bits	S,R	<div> <div>1</div> <div>16</div> <div>17</div> <div>18</div> <div>32</div> <div>MAGNITUDE</div> <div>0</div> <div>MAGNITUDE</div> </div>

Signed Integer Formats
Figure 6-1

Unsigned Integers

Unsigned integers can be 16, 32, or 64 bits long. Regardless of length or addressing mode, all of the bits in the unsigned integer represent the magnitude of the number.

Most operations work for both signed and unsigned numbers. Special unsigned support is provided only for those magnitude branch instructions that allow results to be evaluated as unsigned integers. Multiply and divide instructions do not work correctly for unsigned integers.

Table 6-2 lists the instructions that operate on signed and unsigned integers.

Table 6-2
Integer Arithmetic Instructions

Mnem	Name	Modes	Description
A	Add Fullword	I	Adds the 32-bit contents of a memory location to the contents of R.
A1A	Add 1 to A	S,R,V	Adds one to the contents of A.
A2A	Add 2 to A	S,R,V	Adds two to the contents of A.
ACA	Add CBIT to A	S,R,V	Adds the value of CBIT to the contents of A.
ADD	Add	S,R,V	Adds the contents of a 16-bit memory location to the 16-bit contents of A.
ADL	Add Long	V	Adds the 32-bit contents of a memory location to the 32-bit contents of L.
ADLL	Add LINK to L	V	Adds the value of LINK to the contents of L.
ADLR	Add LINK to R	I	Adds the value of LINK to the contents of R.
AH	Add Halfword	I	Adds the 16-bit contents of a memory location to the contents of r.
C	Compare Fullword	I	Compares the contents of R to the contents of a memory location and sets the condition codes to reflect the result of the compare.
CH	Compare Halfword	I	Compares the contents of r to the contents of a memory location and sets the condition codes to reflect the result of the compare.
CHS	Change Sign	I	Complements bit 1 of R.
CHS	Change Sign	S,R,V	Complements bit 1 of A.
CSA	Copy Sign of A	S,R,V	Sets CBIT to the value of bit 1 in A, then sets bit 1 of A to 0.
CSR	Copy Sign	I	Copies bit 1 of R into CBIT and resets bit 1 of R to 0.
D	Divide Fullword	I	Divides the 64-bit contents of R and R+1 by the 32-bit contents of a memory location.
DAD	Double Add	S,R	Adds the 31-bit contents of a memory location to the 31-bit contents of A and B.
DH	Divide Halfword	I	Divides the 32-bit contents of R by the 16-bit contents of a memory location.
DH1	Decrement r by 1	I	Decrements r by 1 and stores the results in r.
DH2	Decrement r by 2	I	Decrements r by 2 and stores the results in r.

Table 6-2 (continued)
Integer Arithmetic Instructions

Mnem	Name	Modes	Description
DIV	Divide	S,R	Divides the 31-bit contents of A and B by the 16-bit contents of a memory location.
DIV	Divide	V	Divides the 32-bit contents of L by the 16-bit contents of a memory location.
DM	Decrement Memory Fullword	I	Decrements the contents of the specified memory location by 1.
DMH	Decrement Memory Halfword	I	Decrements the contents of the specified memory location by 1.
DR1	Decrement R by 1	I	Decrements R by 1 and stores the result in r.
DR2	Decrement R by 2	I	Decrements R by 2 and stores the result in r.
DSB	Double Subtract	S,R	Subtracts the 31-bit contents of a memory location from the 31-bit contents of A and B.
DVL	Divide Long	V	Divides the 64-bit contents of E and L by the 32-bit contents of a memory location.
IH1	Increment r by 1	I	Increments r by 1 and stores the result in r.
IH2	Increment r by 2	I	Increments r by 2 and stores the result in r.
IM	Increment Memory Fullword	I	Increments the contents of the specified memory location by 1.
IMH	Increment Memory Halfword	I	Increments the contents of the specified memory location by 1.
IR1	Increment R by 1	I	Increments R by 1 and stores the result in R.
IR2	Increment R by 2	I	Increments R by 2 and stores the result in R.
M	Multiply Fullword	I	Multiplies the 32-bit contents of R by the 32-bit contents of a memory location to get a 64-bit result.
MH	Multiply Halfword	I	Multiplies the 16-bit contents of r by the 16-bit contents of a memory location to get a 32-bit result.

Table 6-2 (continued)
Integer Arithmetic Instructions

Mnem	Name	Modes	Description
MPL	Multiply Long	V	Multiplies the 32-bit contents of L by the 32-bit contents of a memory location to get a 64-bit result.
MPY	Multiply	S,R	Multiplies the 16-bit contents of A by the 16-bit contents of a memory location to get a 31-bit result.
MPY	Multiply	V	Multiplies the 16-bit contents of A by the 16-bit contents of a memory location to get a 32-bit result.
MPY	Multiply	I	Multiplies the 16-bit contents of r by the 16-bit contents of a memory location to get a 32-bit result.
PID	Position for Integer Divide	S,R	Converts the 16-bit integer in A to a 31-bit integer in A and B.
PID	Position for Integer Divide	I	Converts the 32-bit integer in R to a 64-bit integer in R and R+1.
PIDA	Position for Integer Divide	V	Converts the 16-bit integer in A to a 31-bit integer in L.
PIDH	Position for Integer Divide	I	Converts the 16-bit integer in r to a 32-bit integer in R.
PIDL	Position for Integer Divide	V	Converts the 32-bit integer in L to a 64-bit integer in L and E.
PIM	Position After Integer Multiply	S,R	Converts the 31-bit integer in A and B to a 16-bit integer in A.
PIM	Position After Integer Multiply	I	Converts the 64-bit integer in R and R+1 to a 32-bit integer in R.

Table 6-2 (continued)
Integer Arithmetic Instructions

Mnem	Name	Modes	Description
PIMA	Position After Integer Multiply	V	Converts the 32-bit integer in L to a 16-bit integer in A.
PIMH	Position for Integer Multiply	I	Converts the 32-bit integer in R to a 16-bit integer in r.
PIML	Position After Integer Multiply Long	V	Converts the 64-bit integer in L and E to a 32-bit integer in L.
S1A	Subtract 1 From A	S,R,V	Subtracts 1 from the contents of A.
S2A	Subtract 2 From A	S,R,V	Subtracts 2 from the contents of A.
S	Subtract Fullword	I	Subtracts the 32-bit contents of a memory location from the 32-bit contents of R.
SBL	Subtract Long	V	Subtracts the 32-bit contents of a memory location from the 32-bit contents of L.
SH	Subtract Halfword	I	Subtracts the 16-bit contents of a memory location from the 16-bit contents of r.
SSM	Set Sign Minus	S,R,V	Sets bit 1 of A to 1.
SSM	Set Sign Minus	I	Sets bit 1 of R to 1.
SSP	Set Sign Plus	S,R,V	Sets bit 1 of A to 0.
SSP	Set Sign Plus	I	Sets bit 1 of R to 0.
SUB	Subtract	S,R,V	Subtracts the 16-bit contents of a memory location from the 16-bit contents of A.
TCA	Two's Complement A	S,R,V	Forms the two's complement of the contents of A.
TCL	Two's Complement L	V	Forms the two's complement of the contents of L.

Table 6-2 (continued)
Integer Arithmetic Instructions

Mnem	Name	Modes	Description
TC	Two's Complement R	I	Forms the two's complement of the contents of R.
TCH	Two's Complement r	I	Forms the two's complement of the contents of r.
TM	Test Memory Fullword	I	Tests the contents of a memory location and sets the condition codes to reflect the result of the test.
TMH	Test Memory Halfword	I	Tests the contents of a memory location and sets the condition codes to reflect the result of the test.

Addresses

The 50 Series processors manipulate addresses as if they were unsigned integers. Table 6-3 lists the instructions that handle addresses.

Table 6-3
Address Manipulation Instructions

Mnem	Name	Modes	Description
EAFA	EA to FAR	V,I	Calculates an effective address and loads it into the specified FAR.
FLX, DFLX, QFLX	Load Floating Index	R,V	Loads X with a multiple of the contents of a memory location.
CEA	Compute EA	S,R	Uses the contents of A as an indirect address, calculates an effective address from the referenced location and loads the EA into A.
EAA	Effective Address to A	R	Loads an effective address into A.
EAL	Effective Address to L	V	Loads an effective address into L.
EALB	Effective Address to LB	V,I	Loads an effective address into LB.
EAR	Effective Address to R	I	Loads an effective address into R.
EAXB	Effective Address to XB	V,I	Loads an effective address into XB.

Fixed-point Operations

The 50 Series processors can perform several kinds of operations on fixed-point data. Some examples are setting or resetting a single bit in a logical value, or storing an unsigned integer into a memory location. Table 6-4 lists the instructions that move fixed-point data from one place to another. Table 6-5 describes a group of special load/store instructions. Table 6-6 lists the instructions that shift the contents of a 16-bit or 32-bit register. Table 6-7 shows instructions that can be used to set or reset all or part of a piece of data.

Table 6-4
Data Movement Instructions

Mnem	Name	Modes	Description
DLD	Double Load	S,R	Loads A and B with the contents of two 16-bit memory locations.
DST	Double Store	S,R	Stores the contents of A and B into two 16-bit memory locations.
I	Interchange R and Memory Fullword	I	Interchanges the contents of R and a memory location.
IAB	Interchange A and B	S,R,V	Interchanges the values of A and B.
ICA	Interchange Characters in A	S,R,V	Interchanges the contents of the two bytes in A.
ICBL	Interchange and Clear Left	I	Interchanges the contents of the bytes in r, then loads zeroes into the leftmost byte of r.
ICBR	Interchange and Clear Right	I	Interchanges the contents of the bytes in r, then loads zeroes into the rightmost byte of r.
ICHL	Interchange Halfwords and Clear Left	I	Interchanges the contents of bits 1 to 16 and bits 17 to 31 of R, then loads bits 1 to 16 of R with zeroes.
ICHR	Interchange Halfwords and Clear Right	I	Interchanges the contents of bits 1 to 16 and bits 17 to 31 of R, then loads bits 17 to 31 of R with zeroes.
ICL	Interchange and Clear Left	S,R,V	Interchanges the contents of the bytes in A, then loads zeroes into the leftmost byte of A.
ICR	Interchange and Clear Right	S,R,V	Interchanges the contents of the bytes in A, then loads zeroes into the rightmost byte of A.

Table 6-4 (continued)
Data Movement Instructions

Mnem	Name	Modes	Description
IH	Interchange r and Memory	I	Interchanges the contents of r and a memory location.
ILE	Interchange E and L Halfword	V	Interchanges the contents of E and L.
IMA	Interchange A and Memory	S,R,V	Interchanges the contents of A and a memory location.
IRB	Interchange Register Bytes	I	Interchanges the contents of bits 1 to 8 and bits 9 to 16 of r.
IRH	Interchange Register Halves	I	Interchanges the contents of bits 1 to 16 and bits 17 to 32 of R.
L	Load Fullword	I	Loads the contents of a memory location into R.
LDA	Load A	S,R,V	Loads the contents of a memory location into A.
LDL	Load long	V	Loads the contents of a memory location into L.
LDX	Load X	S,R,V	Loads the contents of a memory location into X.
LDY	Load Y	V	Loads the contents of a memory location into Y.
LH	Load Halfword	I	Loads the contents of a memory location into r.
LHL1	Load Halfword Left Shifted By 1	I	Shifts the contents of a memory location left one bit and loads the result into r.
LHL2	Load Halfword Left Shifted By 2	I	Shifts the contents of a memory location left two bits and loads the result into r.
LHL3	Load Halfword Left Shifted By 3	I	Shifts the contents of a memory location left three bits and loads the result into r.
ST	Store Fullword	I	Stores the contents of R into a memory location.
STA	Store A	S,R,V	Stores the contents of A into memory.

Table 6-4 (continued)
Data Movement Instructions

Mnem	Name	Modes	Description
STAC	Store A Conditionally	V	Stores the contents of A into memory if the contents of the specified memory location equal the contents of B.
STCD	Store Conditional Fullword	I	Stores the contents of R into the location specified by EA if the contents of R+1 equal the contents of the location specified by EA.
STCH	Store Conditional Halfword	I	Stores the contents of r into the location specified by EA if the contents of bits 17 to 32 equal the contents of the location specified by EA.
STH	Store Halfword	I	Stores the contents of r into a memory location.
STL	Store Long	V	Stores the contents of L into memory.
STLC	Store L Conditionally	V	Stores the contents of L into memory if the contents of the specified memory location equal the contents of E.
STX	Store X	S,R,V	Stores the contents of X into memory.
STY	Store Y	V	Stores the contents of Y into memory.
TAB	Transfer A to B	V	Transfers the contents of A into B.
TAX	Transfer A to X	V	Transfers the contents of A into X.
TAY	Transfer A to Y	V	Transfers the contents of A into Y.
TBA	Transfer B to A	V	Transfers the contents of B into A.
TXA	Transfer X to A	V	Transfers the contents of X into A.
TYA	Transfer Y to A	V	Transfers the contents of Y into A.
XCA	Exchange and Clear A	S,R,V	Exchanges the contents of A and B, then loads zeroes into A.
XCB	Exchange and Clear B	S,R,V	Exchanges the contents of B and A, then loads zeroes into B.

Table 6-5
Special Load/Store Instructions

Mnem	Name	Modes	Description
RSAB	Save Registers	V,I	Saves the contents of the general, floating, temporary, and base registers in a block of consecutive memory locations.
RRST	Restore Registers	V,I	Restores the values of the general, floating, temporary, and base registers with information contained in a block of consecutive memory locations.
LDAR	Load Addressed Register	I	Loads the contents of a register file location into R.
LDLR	Load L from Register File	V	Loads the contents of a register file location into L.
STAC	Store A Conditionally	V	Stores the contents of A at the specified address if the contents of the specified address are equal to the contents of B.
STAR	Store Addressed Register	I	Stores the contents of the specified R in a register file location.
STLC	Store L Conditionally	V	Stores the contents of L into the specified address if the contents of the specified address are equal to the contents of E.
STLR	Store L Into Register File	V	Loads the contents of L into a register file location.

Table 6-6
Shift Instructions

Mnem	Name	Modes	Description
ALL	A Left Logical	S,R,V	Shifts the contents of A left a specified number of bits.
ALR	A Left Rotate	S,R,V	Shifts the contents of A left a specified number of bits, rotating bit 1 into bit 16.
ALS	A Left Shift	S,R,V	Shifts the contents of A left a specified number of bits.
ARL	A Right Logical	S,R,V	Shifts the contents of A right a specified number of bits.
ARR	A Right Rotate	S,R,V	Shifts the contents of A right a specified number of bits, rotating bit 16 into bit 1.
ARS	A Right Shift	S,R,V	Shifts the contents of A right a specified number of bits.
LLL	L Left Logical	S,R,V	Shifts the contents of L left a specified number of bits.
LLR	L Left Rotate	S,R,V	Shifts the contents of L left a specified number of bits, rotating bit 1 into bit 16.
LLS	L Left Shift	S,R	Shifts the contents of A and B left a specified number of bits, bypassing bit 1 of B.
LLS	L Left Shift	V	Shifts the contents of L left a specified number of bits.
LRL	L Right Logical	S,R,V	Shifts the contents of L right a specified number of bits.
LRR	L Right Rotate	S,R,V	Shifts the contents of L right a specified number of bits, rotating bit 16 into bit 1.
IRS	L Right Shift	V	Shifts the contents of L right a specified number of bits.
IRS	L Right Shift	S,R	Shifts the contents of A and B right a specified number of bits, bypassing bit 1 of B.
ROT	Rotate	I	Rotates the contents of R a specified number of bits in a specified direction.
SHA	Arithmetic Shift	I	Shifts the contents of R a specified number of bits in a specified direction.
SHL	Logical Shift	I	Shifts the contents of R a specified number of bits in a specified direction.
SL1	Shift R Left 1	I	Shifts the contents of R left one bit.
SL2	Shift R Left 2	I	Shifts the contents of R left two bits.

Table 6-6 (continued)
Shift Instructions

Mnem	Name	Modes	Description
SR1	Shift R Right 1	I	Shifts the contents of R right one bit.
SR2	Shift R Right 2	I	Shifts the contents of R right two bits.
SHL1	Shift r Left 1	I	Shifts the contents of r left one bit.
SHL2	Shift r Left 2	I	Shifts the contents of r left two bits.
SHR1	Shift r Right 1	I	Shifts the contents of r right one bit.
SHR2	Shift r Right 2	I	Shifts the contents of r right two bits.

Notes to Table 6-6

The instructions in Table 6-6 specify three types of shift operations. An instruction that performs a logical shift treats the data to be shifted as a logical string of bits, shifting zeroes into the vacated bits. The CBIT and LINK reflect the state of the last bit shifted out.

An instruction performing an arithmetic shift treats the data as a signed number. For a right arithmetic shift, the instruction shifts in copies of the sign bit into the vacated bits; the CBIT and LINK reflect the state of the last bit shifted out. For a left arithmetic shift, the instruction shifts zeroes into the vacated bits. If there is a sign change in bit 1 (interpreted as an overflow condition), an integer exception occurs. (See Chapter 10.)

An instruction that performs a rotate shifts bits out of one side of the data word and loads them into vacated bits on the other side. The CBIT and LINK contain a copy of the last bit rotated.

Table 6-7
Clear Register/Memory Instructions

Mnem	Name	Modes	Description
CAL	Clear A Left Byte	S,R,V	Sets bits 1 to 8 of A to 0.
CAR	Clear A Right Byte	S,R,V	Sets bits 9 to 16 of A to 0.
CR	Clear Register	I	Sets the specified register to 0.
CRA	Clear A	S,R,V	Resets the contents of A to 0.
CRB	Clear B	S,R,V	Resets the contents of B to 0.
CRBL	Clear High Byte 1 Left	I	Sets bits 1 to 8 of the specified register to 0.
CRBR	Clear High Byte 1 Right	I	Sets bits 9 to 16 of the specified register to 0.
CRE	Clear E	V	Resets the contents of E to 0.
CRHL	Clear Left Halfword	I	Sets bits 1 to 16 of the specified register to 0.
CRHR	Clear Right Halfword	I	Sets bits 17 to 32 of the specified register to 0.
CRL	Clear L	S,R,V	Resets the contents of L to 0.
CRLE	Clear L and E	V	Resets the contents of L and E to 0.
ZM	Zero Memory Fullword	I	Resets the 32-bit contents of the specified memory location to 0.
ZMH	Zero Memory Halfword	I	Resets the 16-bit contents of the specified memory location to 0.

Field Operations

The 50 Series processors support a group of instructions that perform field operations. These instructions use the field address and length registers in their manipulations. These registers are abbreviated as FAR, for field address register, or FLR, for field length register; but both are specified in the same 64-bit register shown in Figure 6-2.

The field address and length registers overlap the floating accumulators as shown in Figures 6-2 and 6-3. The precise format and overlap, however, varies from one Prime machine to another. Table 6-8 lists the field operation instructions.

Table 6-8
Field Operation Instructions

Mnem	Name	Modes	Description
ALFA	Add Long to FAR	V	Adds the contents of L to the contents of the specified FAR.
ARFA	Add R to FAR	I	Adds the contents of the specified R to the contents of the specified FAR.
EAFA	EA to FAR	V,I	Calculates an effective address and loads it into the specified FAR.
LDC	Load Character	V,I	Calculates an effective address. Loads the character in the specified field into the addressed location.
LFLI	Load Immediate to FLR	V,I	Loads an immediate value into the specified FLR.
STFA	Store FAR	V,I	Calculates an effective address and stores the contents of the specified FAR into the addressed location.
STC	Store Character into Field	V,I	Stores the contents of a register into the specified field.
TFLI	Transfer Long from FLR	V	Transfers the contents of the specified FLR to L.
TLFL	Transfer Long to FLR	V	Transfers the contents of L into the specified FLR.
TFLR	Transfer FLR to R	I	Transfers the contents of the specified FLR to the specified R.
TRFL	Transfer R to FLR	I	Transfers the contents of the specified R to the specified FLR.

1	2	3	4	5	16	17	32	33	36	37	43	44	64
0	RING	0	SEGMENT		OFFSET		BIT		0000000		LENGTH		

Bits	Mnem	Description
2 to 3	RING	Specifies the ring number of the field address.
5 to 16	SEGMENT	Specifies the segment number of the field address.
17 to 32	OFFSET	Specifies the offset number of the field address.
33 to 36	BIT	Specifies the bit number of the field address.
37 to 43	---	Must be 0.
44 to 64	LENGTH	Specifies 21 bits of field length.

Format of Field Address and Length Register (FAR, FLR)
Figure 6-2

1	48	49	64
	DOUBLE PRECISION FRACTION		EXP

Bits	Mnem	Description
1 to 48	DOUBLE PRECISION FRACTION	Specifies the sign and magnitude of a floating-point number.
49 to 64	EXP	Specifies the exponent of a floating-point number.

Format of Floating Register (F)
Figure 6-3

FLOATING-POINT NUMBERS

Floating-point numbers are made up of two fields:

- A fraction containing the two's complement value of the number
- An exponent

Bits 1 to 24 (single precision), bits 1 to 48 (double precision), or bits 1 to 48 and bits 65 to 112 (quad precision, not applicable to the earlier processors listed on page 1-1) contain the two's complement value representing the fraction of the number. Bit 1 indicates whether the number is positive (bit 1 contains 0) or negative (bit 1 contains 1). The binary point lies between bits 1 and 2.

Bits 25 to 32 (single precision) or bits 49 to 64 (double and quad precision) contain the exponent of the floating-point number. The exponent, in excess 128 form, is the power of 2 that is to multiply the fraction. The true value of the exponent is always 128 less than the value contained in the exponent field.

In other words:

$$\text{Floating-point Number} = (\text{fraction}) * (2^{(\text{exponent}-128)})$$

Figure 6-4 shows the format of single (SP), double (DP), and quad precision (QP) numbers. The abbreviated names of the SP, DP, and QP floating-point accumulators are FAC, DAC, and QAC, respectively. The number of floating accumulators for each mode and precision type appears in Table 6-9. These accumulators are overlapped, sharing the same storage.

Table 6-9
Number of Floating-point Accumulators

Name	R Mode	V Mode	I Mode
FAC	1	1	2
DAC	1	1	2
QAC	None	1	1

Location	Size	Format
Memory	Single Precision	<div>1 24 25 32</div> <div>FRACTION EXPONENT</div>
Memory	Double Precision	<div>1 48 49 64</div> <div>FRACTION EXPONENT</div>
Memory	Quad Precision	<div>1 48 49 64</div> <div>FRACTION EXPONENT</div>
		<div>65 112 113 128</div> <div>FRACTION UNUSED</div>
Accumulator	Single Precision	<div>1 48 49 64*</div> <div>FRACTION EXPONENT</div>
Accumulator	Double Precision	<div>1 48 49 64</div> <div>FRACTION EXPONENT</div>
Accumulator	Quad Precision	<div>1 48 49 64</div> <div>FRACTION EXPONENT</div>
		<div>65 112 113 128</div> <div>FRACTION UNUSED</div>

* The format of the FAC for the earlier systems (listed on page 1-1) appears in Appendix B.

Floating-point Formats
Figure 6-4

Floating Accumulators

In R and V modes, FAC or DAC occupies locations '12 to '13 in the current register file set. I mode has two FAC or DAC accumulators labeled 0 and 1 that occupy locations '10 to '13. For all modes, QAC combines floating accumulators 0 and 1 into one accumulator occupying locations '10 to '13. The high-order fraction bits and the exponent of a quad precision floating-point number are found in DAC1 in I mode.

The field address and length registers overlap the floating-point registers. Using FAR0, FLR0, and FAC0 instructions will not modify the contents of FAC1, FLR1, or FAC1, and vice versa. However, mixing FAR0 and FLR0 instructions with FAC0 (32I mode), or combining FAR1 or FLR1 instructions with FAC1 (32I mode or FAC 64V mode), produces variable results from machine to machine and attempt to attempt.

There is no particular implied overlap amongst LLR and SLR instructions. Extracting the exponent can best be done with either an LDA 6 (address trap) or a DFST T followed by an LDA T+3.

Floating-point Operations

In R, V, and I modes, floating-point has instructions that operate from memory to register or on a register alone. I mode also has some floating-point instructions that operate in a register to register and immediate fashion. Table 6-10 lists all floating-point operations. The first letter of a floating-point instruction shows its data type:

- F for single precision
- D for double precision
- Q for quad precision

Table 6-10
Floating-point Instructions

Mnem	Name	Modes
FAD, DFAD QFAD FA, DFA, QFA	Floating Add	R, V V I
FC, DFC, QFC	Floating Compare	I
FCM, DFCM QFCM	Floating Complement	R, V, I V, I
FCS, DFCS QFCS	Floating Compare and Skip	R, V V
FDV, DFDV QFDV FD, DFD, QFD	Floating Divide	R, V V I
FLD, DFLD QFLD FL, DFL, QFL	Floating Load	R, V V I
FMP, DFMP QFMP FM, DFM, QFM	Floating Multiply	R, V V I
FSB, DFSB QFSB FS, DFS, QFS	Floating Subtract	R, V V I
FST, DFST QFST	Floating Store	R, V, I V, I

Manipulating Floating-point Numbers

The following topics are pertinent for many operations since they deal with some aspect of handling the accumulator results: overflow or underflow, normalization, zero, and rounding.

Overflow and Underflow: Overflow occurs when the number of bits in the exponent of a result exceeds the capacity of its destination's exponent. Underflow happens when the exponent of a result is too small to be represented in a specified register or memory location. For all 50 Series systems, upon overflow or underflow, the fraction is incorrect and the exponent has the incorrect sign. Underflow can be distinguished from overflow by checking the sign of the exponent.

A floating-point exception occurs upon overflow or underflow. When this happens the processor checks the content of bit 7 of the keys for the prescribed action. If bit 7 contains 1, the processor merely sets CBIT to 1. If bit 7 contains 0, the processor sets CBIT to 1 and also loads the FADDR, FCODEH, and FCODEL registers of the user register file as described in Chapter 10.

Because the FAC has a much greater exponent range than the memory format, overflow in single precision is detected only when a store operation is performed. This situation produces a store exception. See Chapter 10 for more information.

Normalization: All numbers generated by arithmetic floating-point operations are normalized by the processor. A number is defined as being normalized either when bits 1 and 2 contain different values or when the number is a zero with both fraction and exponent equal to zero. If this is not the case when a result is first generated, the processor shifts the fraction to the left and adjusts the exponent appropriately until bits 1 and 2 do have different values.

All systems but the earlier ones (listed on page 1-1) retain two extra least significant bits of precision, called guard bits, that are shifted into the right side of the fraction during the first two left bit shifts. If more bit shifts are needed, the processor shifts in zeroes.

For the earlier systems listed on page 1-1, see Appendix B for information concerning normalization and their guard bits.

Zero: A proper Prime floating-point zero has every bit reset to zero. Any floating-point value, however, that contains a zero fraction (all fraction bits reset to zero) is interpreted as a zero value. Specifically, non-zero exponents are ignored in all operations when the fraction is zero, and such a value is called a "dirty zero".

Rounding: Table 6-11 lists the prerequisites and procedures for rounding. (See Appendix B for rounding on the earlier systems listed on page 1-1.) Rounding is done after the result is normalized; rounding in turn may produce a result that needs to be normalized again.

Table 6-11
Rounding Prerequisites and Procedures*

Type	Operation	Rounding Description
SP	Add, subtract, multiply	In rounding mode (bit 13 of keys is 1), add guard bit 1 to FAC bit 48 and normalize. FRN may be done in rounding mode and a double round will not occur.
	Divide	Always rounds. 49 fraction bits are generated for rounding to 48.
	Store	In rounding mode, add 1 to FAC bit 25, normalize result, but leave original FAC fraction unchanged.
	Compare and Skip	In rounding mode, add 1 to FAC bit 25, normalize result, store in temp register for compare, but do not load back into FAC; original FAC fraction left unchanged.
DP	Arithmetic operations	Rounding is the same as in SP.
	Others	Rounding never done.
QP	Divide	Always rounds. 97 fraction bits are generated for rounding to 96.
	Others	Rounding never done.

*See Appendix B for rounding on the earlier systems listed on page 1-1.

Normalized Versus Unnormalized Operands

Floating-point operations in Prime processors always produce normalized results. Hence, an unnormalized number can only enter the system as an external input operand. Instructions assume normalized floating-point operands; however, no exception results from unnormalized operands apart from those in a divide. To ensure accurate floating-point results, use normalized numbers.

There are several ways of obtaining normalized numbers. FAD, DFAD, or QFAD instructions normalize an unnormalized memory argument when the other value is a floating-point zero (defined as having both fraction and exponent equal to zero). The instruction sequence DFLD, DFCM, and DFCM also normalizes an operand. Data conversion instructions FLOT, FLT, FLTA, and FLTH convert integers to normalized floating-point numbers. Lastly, standard Prime compilers and assemblers produce normalized constants.

Floating-point branch, skip, and logicize (logical test) instructions work correctly on normalized or nearly normalized numbers because these instructions check the first 32 fraction bits only for equal to zero and less than zero. (A normalized number has different values for bits 1 and 2; in a nearly normalized number as defined here, at least one of the examined bits has a different value from the rest, for a positive value.)

When floating-point instructions are performed on unnormalized numbers, the following guarantees apply. The instructions do not hang or deviate from the processor's normal flow of control. Add, subtract, complement, and compare and skip instructions produce approximately correct answers. Bit for bit identical values will compare equal or subtract to zero by using either a subtract instruction, or a complement instruction that is followed by an add. All floating load and store instructions copy 32-bit, 64-bit, or 128-bit quantities from place to place as appropriate without faulting or normalizing unless single precision is used and rounding mode is enabled. Because single precision rounding mode rounds and normalizes on a compare and store, the single precision numbers will always be normalized before a store, causing a bit pattern change.

Using unnormalized numbers for some floating-point operations causes problems in the following cases. Compare and skip instructions fail on machines that look first at the sign, then the exponent, and finally the fraction for possible inequality. Divide produces indeterminate results on all processors but the earlier ones (listed on page 1-1) when confronted with unnormalized numbers. Accuracy loss is probable for all other operations on all other systems.

Programming Notes: FORTRAN 66 programmers often use floating-point to store character strings. To the processor, these character strings are unnormalized floating-point values. REAL*8 values work for copy and identity comparison operations, but make sorted ordering impossible. REAL*4 values work in a similar fashion if rounding mode is not enabled. For storing character strings, use INTEGER*4 since they work faster and permit sorting.

Floating-point Accuracy and Precision

For the earlier systems listed on page 1-1, see Appendix B for discussions and tables concerning their floating-point accuracy and precision.

Table 6-12 shows the accuracy of floating-point arithmetic instructions as performed on normalized numbers. The number of guard bits preserved need be no greater than two to simulate infinite precision if normalized numbers are used and the algorithm is carefully designed.

The values in Table 6-12 refer to the number of fraction bits guaranteed to be accurate for the indicated processor. This number includes the sign bit because the fraction represents a two's complement value. Other manuals may emulate a sign-magnitude representation in statements about accuracy. A sign-magnitude representation requires a 1 to be subtracted from all entries in this table. Worst case normalization is included in all results. The accuracy of an infinite precision result lies closer to the number indicated than to either of its neighboring representations.

Table 6-13 shows floating-point precision for all 50 Series systems when performed with normalized numbers. The degree of floating-point precision and accuracy varies among these systems due to their differences in implementation, as discussed in the following paragraphs.

The fraction values in Table 6-13 refer to the number of fraction bits for the indicated processor. This number includes the sign bit because the fraction represents a two's complement value; other manuals may emulate a sign-magnitude representation. A sign-magnitude representation requires a 1 to be subtracted from all fraction entries in this table.

Table 6-12
Floating-point Instruction Accuracy

Instruction	Accuracy
FAD	48+#
DFAD	48+#
FSB	48+#
DFSB	48+#
FMP	48+#
DFMP	48+#
FDV	48+*
DFDV	48+*
QFAD	96
QFSB	96
QFMP	96
QFDV	96

Notes to Table 6-12

+ means 2 extra guard bits are used.

means rounding mode can be used.

* means rounding is always performed.

See Appendix B for the earlier systems listed on page 1-1.

Table 6-13
Floating-point Precision

Structure	Precision
Fraction Bits:	
Memory	24/48/96
Accumulator	48/48/96
Exponent Bits:	
Memory	8/16/16
Accumulator	16/16/16
Guard Bits	2 for all, excepting quad
Rounds Automatically	For divide regardless of mode or precision. For rest of SP or DP instructions in rounding mode only.

Notes to Table 6-13

The number of fraction and exponent bits is shown in SP/DP/QP form.

See Appendix B for the earlier systems listed on page 1-1.

The following discussion applies only for the 2350 to 9955 II. For the earlier processors listed on page 1-1, see Appendix B.

All SP and DP arithmetic operations generate at least 48 fraction bits plus two guard bits to safeguard accuracy during normalization. If more than two bit shifts are needed during normalization, the processor shifts in zeroes. After normalization, the processor rounds if in rounding mode (as explained in Table 6-13), and then renormalizes the result.

To store the number in SP memory while in non-rounding mode, the processor truncates the result to 24 bits. In rounding mode, the processor rounds the stored value to 24 bits.

Quad precision divide instructions generate 97 fraction bits for rounding to 96. All other operations produce 96 fraction bits of fraction; guard bits are not used.

The quad floating point accumulator and memory is 128 bits long. Bits 1 to 112 of this are used for calculations. Bits 113 to 128 are unused but are subject to the following restrictions. QFLD loads bits 1 to 112 into QAC and zeroes QAC bits 113 to 128, or QFLD loads 128 bits into QAC. QFLD followed by QFST does not reliably copy 128 bits of data. All arithmetic operations zero bits 113 to 128 on completion.

Converting Datatypes

Several 50 Series system instructions convert floating-point numbers to integers and vice versa. Table 6-14 lists these instructions and gives a brief description of each.

Table 6-14
Conversion Instructions

Mnem	Name	Modes	Description
DBLE	Convert Single to Double	I	Converts the single precision floating-point number to a double precision floating-point number.
DRN	Double Round from Quad	V,I	Converts a quad precision floating-point accumulator value to a double precision floating-point number.
DRNM	Double Round from Quad to Minus Infinity	V,I	Converts a quad precision floating-point accumulator value to a double precision floating-point number.
DRNP	Double Round from Quad to Plus Infinity	V,I	Converts a quad precision floating-point accumulator value to a double precision floating-point number.
DRNZ	Double Round from Quad to Zero	V,I	Converts a quad precision floating-point accumulator value to a double precision floating-point number.
FCDQ	Floating Convert Double to Quad	V,I	Converts a double precision floating-point accumulator number to a quad precision floating-point number.
FDEL	Floating Point Convert Single to Double	R,V	Converts a single precision floating-point accumulator number to a double precision floating-point number.
FLOT	Convert Integer to Floating Point	R	Converts the 31-bit contents of A and B to a normalized floating-point number and stores the 31-bit result in the floating accumulator.
FLT	Convert Integer to Floating Point	I	Converts the contents of the specified R to a normalized floating-point number and stores the result in the floating accumulator.
FLTA	Convert Integer to Floating Point	V	Converts the 16-bit contents of A to normalized floating-point number and stores the result in the floating accumulator.
FLTH	Convert Halfword Integer to Floating Point	I	Converts the 16-bit integer contained in the specified r to a normalized floating-point number and stores it in the floating accumulator.
FLTL	Convert Integer to Floating Point	V	Converts the 32-bit contents of L to a floating-point number and stores the result in the floating accumulator.

Table 6-14 (continued)
Conversion Instructions

Mnem	Name	Modes	Description
FRN	Floating Round	R,V,I	Rounds the fraction of a floating-point accumulator number to the nearest 24-bit fraction.
FRNM	Floating Round from DP to Minus Infinity	V,I	Converts a double precision floating-point accumulator value to a single floating-point number.
FRNP	Floating Round from DP to Plus Infinity	V,I	Converts a double precision floating-point accumulator value to a single precision floating-point number.
FRNZ	Floating Round from DP to Zero	V,I	Converts a double precision floating-point accumulator value to a single precision floating-point number.
INT	Convert Floating Point to Integer	R	Converts the DP number in a floating accumulator to a 31-bit integer and stores it in A and B.
INT	Convert Floating Point to Integer	I	Converts the DP number in a floating accumulator to a 32-bit integer and stores it in GR2.
INTA	Convert Floating Point to Integer	V	Converts the DP number in a floating accumulator to a 16-bit integer and stores it in A.
INTH	Convert Floating Point to Halfword Integer	I	Converts the DP number in a floating accumulator to a 16-bit integer and stores it in r.
INTL	Convert Floating Point to Long Integer	V	Converts the DP number in the floating accumulator to a 32-bit number and stores it in L.
QINQ	Floating Convert Integer to Quad	V,I	Converts the truncated integer portion of the floating-point accumulator to a quad precision floating-point number.
QIQR	Floating Convert Integer to Quad Rounded	V,I	Converts the rounded integer portion of the floating-point accumulator to a quad precision floating-point number.

DECIMAL DATA

Decimal data can be represented in packed or unpacked forms.

Unpacked Decimal

There are four forms of unpacked decimal numbers, as shown in Figure 6-5. In this figure, A indicates the ASCII-8 bit that is determined by bit 12 of the keys. (The keys are discussed in Chapter 5.)

Type	Format	Example
Leading Sign, not embedded	First byte contains sign only.	A0101011 A0110011 A0110000 A0110101 + 3 0 5
Trailing Sign, not embedded	Last byte contains sign only.	A0110010 A0110110 A0110001 A0101101 2 6 1 -
Leading Sign, embedded	First byte contains sign and first digit.	A0110110 A0110110 A0111001 A0111001 +6 (6) 6 9 9
Trailing Sign, embedded	Last byte contains sign and last digit.	A0110100 A0110110 A0111000 A1001010 4 6 8 -1 (J)

Unpacked Decimal Formats
Figure 6-5

In the first two cases listed in Figure 6-5, a plus sign represents a positive number, and a minus sign a negative number. You can use a space character to represent a positive sign, and the processor will interpret it correctly. Numerical operations, however, cannot produce positive numbers that contain a space character.

In the two cases where the sign is embedded, a single character represents the appropriate sign and digit. Table 6-15 shows the characters that you use to represent sign/digit combinations.

Table 6-15
Sign/Digit Representations for Unpacked Decimal

Digit	Positive Rep.	Negative Rep.
0	0, space, +, {	}, -
1	1, A	J
2	2, B	K
3	3, C	L
4	4, D	M
5	5, E	N
6	6, F	O
7	7, G	P
8	8, H	Q
9	9, I	R

There are several multiple representations listed above. The processor recognizes all of the representations, but it generates only the first character as the result of an operation. For example, the processor will generate a } to represent a negative zero with embedded sign.

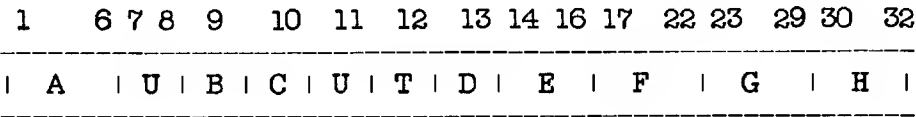
Packed Decimal

The fifth way to represent decimal numbers is called packed decimal. A number in this form uses four bits to represent each digit in the number; the last four bits of the number represent the sign. (Packed decimal numbers are always in trailing sign format.) A decimal number must contain an odd number of digits (excluding the sign digit). It must also begin on a byte boundary.

The sign digit of a decimal result contains a hex C if the sign is positive or a hex D if it is negative. The processor interprets the sign digits of a decimal operand as positive if it contains anything other than a hex C or D.

Control Word Format

Unlike the instructions already listed in this chapter, decimal arithmetic instructions require more information to execute than they can contain. They require a control word to specify the characteristics of the operations to be performed. When a decimal instruction is executing in V mode, L contains a copy of the control word; in I mode, General Register 2 contains the copy. Figure 6-6 shows the format of the control word. Within this figure, F1 and F2 stand for field 1 and field 2, respectively.



Field	Bits	Contents or Meaning
A	1 to 6	Number (0 to '77) of digits in F1
U	7 to 8	Unused; must be zero
B	9	Sign of F1: B=1: sign of F1 is inverse of specified value 0: sign of F1 is as specified
C	10	Sign of F2: C=1: sign of F2 is inverse of specified value 0: sign of F2 is as specified
U	11	Unused; must be zero
T	12	Sign of result: T=1: result is forced positive 0: instruction operation dictates the sign
D	13	Round flag (used only by XMV)
E	14 to 16	Decimal data type of F1
F	17 to 22	Number (0 to '77) of digits in F2
G	23 to 29	Scale differential
H	30 to 32	Decimal data type of F2

Decimal Control Word Format
Figure 6-6

Most of the fields are self-explanatory. Fields D, E, G, H, and T, however, merit extra discussion.

Field D is used only by the XMV instruction. This field tells the processor whether to round the decimal number in F1 or not. If D contains a 0, no rounding occurs. If D contains 1 and the scale differential, the G field, is positive, rounding is performed. The rounding is accomplished by adding a 1 to the (scaled) results field if the last digit scaled over is 5 or greater when XMV moves the contents of F1 into F2.

Control word fields E and H specify the decimal data types of the operands. Table 6-16 lists the available data types and the codes used to represent them in the control word fields.

Table 6-16
Decimal Data Types

Code	Decimal Data Type
0	Leading separate
1	Trailing separate
3	Packed decimal
4	Leading embedded
5	Trailing embedded

Control word field G specifies the scale differential.

For XAD, XCM, and XMV, G specifies the difference in magnitude between the operators of an instruction. This field contains a 7-bit, two's complement number with the value:

$$F_x = \text{magnitude}(F1) - \text{magnitude}(F2).$$

If F_x is positive, then F1 must be shifted right so that it aligns with F2; if negative, F1 must be shifted left to be aligned with F2.

For example, suppose F1 contains 999V99, and F2 contains 999. The scale differential for these operands would be +2, since F1 must be shifted to the right two digits to align with F2.

For XMP, the scale differential is the length of the multiplicand.

The T bit is used by the decimal instructions XAD, XDV, XMP, and XMV. For all these instructions, results are forced positive if the T bit contains 1.

The descriptions of the decimal instructions (see the Instruction Sets Guide) list the control word fields required for instruction execution. Any unused fields must contain zeroes for proper execution to occur.

Decimal Operations

Decimal results are correct for all the digits shown in the result field. The processor calculates the result to all its bits of precision, then loads as many as can fit into the result field. If the portion stored does not contain the most significant bits of the result, an overflow occurs that causes a decimal exception. (See Chapter 10.)

Register Use

In general, all decimal instructions use GR0, GR1, GR3, GR4, and GR6 in both V and I modes. On the 6350 and 9750 to 9955 II, all decimal instructions use L (GR2 in I mode), FAR0, and FAR1. XDIB and XBTD do not use FAR1, but also use GR4.

Table 6-17 lists the decimal instructions.

Table 6-17
Decimal Instructions

Mnem	Name	Modes	Description
XAD	Decimal Add	V,I	Adds the contents of two decimal fields together and stores the result in the destination field.
XMV	Decimal Move	V,I	Moves the contents of the source field into the destination field.
XCM	Decimal Compare	V,I	Compares the contents of the source and destination fields and sets the condition codes depending on the outcome of the compare.
XMP	Decimal Multiply	V,I	Multiplies the contents of the source and destination fields and stores the result in the destination field.
XDV	Decimal Divide	V,I	Divides the contents of the destination field by the contents of the source field and stores the result and the remainder in the destination field.
XBTD	Binary to Decimal Conversion	V,I	Converts a binary number contained in a register to a decimal number and stores the result in a memory location.
XDTB	Decimal to Binary Conversion	V,I	Converts a decimal number in memory to a binary number and stores the result in a register.
XED	Decimal Edit	V,I	Edits a decimal string under control of an edit subprogram.

CHARACTER STRINGS

Character strings are made up of bytes, with each byte representing one ASCII character. A character string can contain from 1 to 2**17 bytes. Table 6-18 lists the character instructions.

Table 6-18
Character Instructions

Mnem	Name	Modes	Description
LDC	Load Character	V,I	Calculates an effective address. Loads the character in the specified field into bits 9 to 16 of a register. Clears bits 1 to 8.
STC	Store Character	V,I	Stores the contents of bits 9 to 16 of A into the specified field.
ZCM	Compare Character Fields	V,I	Compares two character fields and sets the condition codes depending on the outcome of the compare.
ZED	Edit Character Fields	V,I	Moves characters from one field to another under control of an edit subprogram.
ZFIL	Fill Field	V,I	Stores a character into each byte of the specified field.
ZMV	Move Characters	V,I	Moves characters from one field to another.
ZMVD	Move Equal Length	V,I	Moves characters from one field to another of equal length.
ZTRN	Translate Character Field	V,I	Uses one field to reference a translation table and construct a second field.

The Z-prefix character instructions (that is, all character instructions except LDC and STC) move data in the source string starting from the lowest addressed byte (ascending order). ZED and ZTRN move one byte at a time; ZCM, ZFIL, ZMV, and ZMVD always move four bytes at a time (unless there are fewer than six bytes to move and the source and destination are not aligned).

The Z-prefix character instructions may produce unexpected results if the source and destination strings overlap. For example, suppose ZMV is to move the contents of a large source string into a destination string. Figure 6-7 shows how the source and destination strings overlap; S represents the first byte in the source string (labelled 6) and D represents the first byte in the destination string (labelled 1).

After ZMV moves the first four characters, the strings are as shown in the second part of Figure 6-7. The last part shows how the second move affects the string. The third and subsequent moves would work in the same way. In this case ZMV simply moves all characters in the source string into the destination string straightforwardly, without deviation.

1	2	3	4	5	6	7	8	9	10	11	12	13	Strings before move
A	B	C	D	E	F	G	H	I	J	K	L	M	
^					^								
D					S								After first move
1	2	3	4	5	6	7	8	9	10	11	12	13	
F	G	H	I	E	F	G	H	I	J	K	L	M	
^					^								After second move
D					S								
1	2	3	4	5	6	7	8	9	10	11	12	13	
F	G	H	I	J	K	L	M	I	J	K	L	M	
^					^								
D					S								

Overlapping String Manipulation
Figure 6-7

Suppose, however, that the starting addresses of the two strings are switched. The first five bytes in the source string will be correctly moved, but the rest of the string will have been overwritten by copies of the first five bytes. These same five characters will propagate through the rest of the destination string, as shown in Figure 6-8.

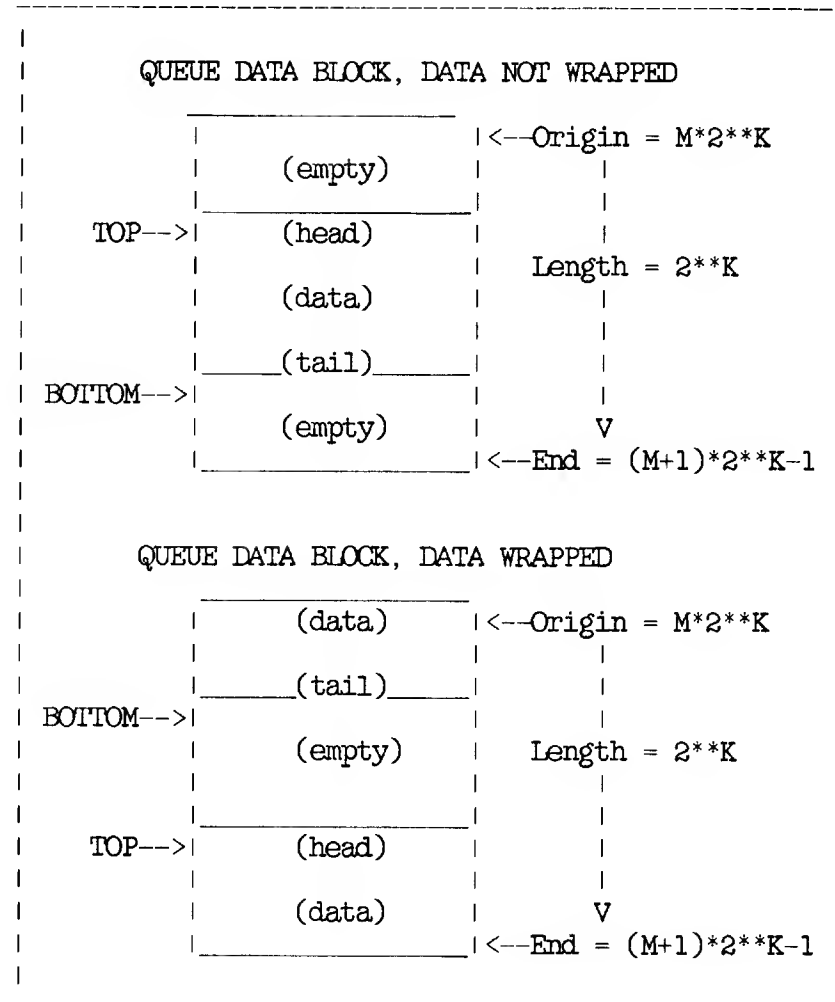
1	2	3	4	5	6	7	8	9	10	11	12	Strings before move
A	B	C	D	E	F	G	H	I	J	K	L	
^					^							
S					D							After first move
1	2	3	4	5	6	7	8	9	10	11	12	
A	B	C	D	E	A	B	C	D	J	K	L	
^					^							After second move
S					D							
1	2	3	4	5	6	7	8	9	10	11	12	
A	B	C	D	E	A	B	C	D	E	A	B	
^					^							
S					D							

String Manipulation
Figure 6-8

While the move shown in Figure 6-8 is useful, it may not be the action that was intended. Overlapping strings produce arbitrarily different results for each Prime machine. For this reason, avoid using overlapping strings in any situation.

QUEUES

A queue is a fixed length, double-ended, circular word buffer. Figure 6-9 shows the format of a typical queue with wrapped and unwrapped data.



Queues With Wrapped and Unwrapped Data
Figure 6-9

QCBs

Each queue in the system is controlled by a queue control block (QCB). This QCB contains information about the queue's size and location in memory, as well as data used to manipulate the elements. In addition, the QCB defines the queue's type. If the value of the QCB's V bit is 0, then the QCB contains physical addresses and the associated queue is called a physical queue. These types of queues are the only ones used

for DMQ operations. If the value of the QCB's V bit is 1, then the QCB contains a virtual segment number and offset rather than a physical address, and the queue is called a virtual queue. Queues of this type are never used for I/O operations.

Try to align QCBs on 8-byte boundaries. DMQ operations (discussed in Chapter 11) require this alignment. For program queue manipulation via the queue instructions, alignment is not necessary but does produce faster queue operations.

Figure 6-10 Shows the format of the QCB.

1		TOP POINTER		16
17		BOTTOM POINTER		32
33		V 000		HIGH ORDER ADDRESS 48
49		SIZE MASK		64

Bits	Name	Description
1 to 16	Top Pointer	Points to first filled location (the head) in the queue.
17 to 32	Bottom Pointer	Points to one past the last filled location (the tail) in the queue.
33	V	Virtual/physical control bit: 0 = physical queue, 1 = virtual queue.
34 to 36	-----	Reserved; must be 0.
37 to 48	High Order Address	High order queue address (if V = 0) or segment number (if V = 1).
49 to 64	Size Mask	Mask; value = (2**K)-1.

Figure 6-10
Format of the QCB

When addressing a QCB, the ring number in the reference specifies the access privileges that will govern the reference. Physical queues can only be accessed from Ring 0.

Queue Specifications

A queue must be $2^{**}K$ words long, where K is an integer between 4 and 16 inclusive. In addition, the queue's starting address must be $M(2^{**}K)$, where M is an integer value. These restrictions allow the firmware to easily identify and locate a queue. Two queues in the system do not have to have the same K in common.

The 50 Series processors use a mask word to add elements to or delete elements from a queue. This mask specifies the size of the queue, and is 16 bits wide. The least significant K bits contain 1 and all other bits contain 0. This means that the numerical value of the mask is $(2^{**}K)-1$. Figure 6-11 contains an example of calculating a mask.

```

-----
| Suppose K = 5.                               |
|      mask = 0000000000011111                |
|      = '37                                   |
|      = 31 decimal                           |
|      = (2**5)-1, QED.                       |
|-----

```

Calculating a Mask
Figure 6-11

The mask also makes it easy to determine the starting and ending addresses of the queue. If P is a pointer to some location within a queue, the address of the queue's origin is:

$$\text{origin} = P \text{ AND } (\text{NOT mask})$$

and the address of the queue's last location is:

$$\text{end} = P \text{ OR mask.}$$

Figure 6-12 contains an example of calculating the starting and ending addresses of a queue.

```

| Suppose K = 5, P = '204, and M = 4.
| mask = '37 and queue length = 2**5 = '37.
|
| origin = '204 AND (NOT '37)
|         = 10000100 AND 1111111111100000
|         = 10000000
|         = '200
|         = 128 decimal
|         = 4(2**5), QED.
|
| end = '204 OR '37
|      = 10000100 OR 11111
|      = 10011111
|      = '237
|      = queue origin + queue length
|      = '200 + '37, QED.

```

Calculating the Origin and End of a Queue
Figure 6-12

Queues operate under one final restriction. They are defined to be empty when the contents of the top pointer equal the contents of the bottom pointer. This means that the maximum number of elements in a queue is $(2**K)-1$.

Queue Algorithms

The 50 Series processors use four algorithms to insert or delete queue elements (depending on the specified operation). Table 6-19 shows the algorithms used for specific operations. The symbols T1 to T5 represent temporary storage registers.

Table 6-19
Queue Algorithms

Inst	Algorithm
RTQ and DMQ Output (I/O)	T1 <- TOP T2 <- BOTTOM If T1 = T2 then A <- 0 CC <- EQ else T3 <- SEGMENT T4 <- MASK A <- (SEGMENT T1) (16 bits) TOP <- T1 AND NOT T4 OR (T1 + 1) AND T4
ABQ and DMQ Input (I/O)	T1 <- TOP T2 <- BOTTOM T3 <- SEGMENT T4 <- MASK T5 <- T2 AND NOT T4 OR (T2 + 1) AND T4 If T1 = T5 then CC <- EQ else location(SEGMENT T2) <- A BOTTOM <- T5
ATQ	T1 <- TOP T2 <- BOTTOM T3 <- SEGMENT T4 <- MASK T1 <- T1 AND NOT T4 OR (T1 - 1) AND T4 If T1 = T2 then CC <- EQ else location(SEGMENT T1) <- A TOP <- T1
RBQ	T1 <- TOP T2 <- BOTTOM If T1 = T2 then A <- 0 CC <- EQ else T3 <- SEGMENT T4 <- MASK T2 <- T2 AND NOT T4 OR (T2 - 1) AND T4 A <- (SEGMENT T2) (16 bits) BOTTOM <- T2

The instructions provided for programmed queue manipulation are shown in Table 6-20. The pointer in the instructions references the QCB for that queue. An RTQ instruction is equivalent to a DMQ output operation, and an ABQ is equivalent to a DMQ input, as noted in Chapter 11, INPUT-OUTPUT.

Table 6-20
Queue Instructions

Mnem	Name	Description
RTQ and DMQ	Remove from Top of Queue	Removes a 16-bit quantity from the top of a queue and places it in r (I mode) or A (other modes)
RBQ	Remove from Bottom of Queue	Removes a 16-bit quantity from the bottom of a queue and places it in r (I mode) or A (other modes).
ABQ and DMQ	Add to the Bottom of Queue	Adds the contents of r (I mode) or A (other modes) to the bottom of the specified queue.
ATQ	Add to the Top of Queue	Adds the contents of r (I mode) or A (other modes) to the top of the specified queue.
TSTQ	Test Queue	Sets r (I mode) or A (other modes) to the number of items in a specified queue and sets the condition codes depending on the new value of r or A.

SUMMARY OF DATATYPES AND APPLICABLE INSTRUCTIONS

Tables 6-21 and 6-22 summarize the different datatypes and list the various operations available. The body of each table shows which instructions perform a specific operation on a specific datatype. For detailed information about each instruction, refer to the instruction dictionaries in the Instruction Sets Guide.

In Table 6-21, the variable aa represents the set of arithmetic conditions [EQ, GE, GT, LE, LT, NE]. Also, Tables 6-21 and 6-22 do not include instructions that operate on CBIT, LINK, the condition codes, or queues.

Table 6-21
Datatypes Summary and Instructions in S, R, and V Modes

Operation	Size of Datatype (in Bits)							
	16 (A)	31 (A/B)	32 (L)	64 (L/E)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Load from memory	LDA	DLD	LDL		FLD	DFLD	QFLD	XMV
Store to memory	STA	DST	STL		FST	DFST	QFST	
Add	ADD	DAD	ADL		FAD	DFAD	QFAD	XAD
Subtract	SUB	DSB	SBL		FSB	DFSB	QFSB	XAD
Multiply	MPY		MPL		FMP	DFMP	QFMP	XMP
Divide	DIV		DVL		FDV	DFDV	QFDV	XDV
Increment	IRS, A1A, A2A							
Decrement	S1A, S2A							
AND	ANA		ANL					
OR	ORA							
XOR	ERA		ERL					
Complement	CMA							
Compare	CAS, CAZ		CLS		FCS	DFCS	QFCS	XCM
Logical test	Laa		LLaa		LFaa	LFaa		
Branch	Baa		BLaa		BFaa	BFaa		
Logical left shift	ALL		LLL					
Logical right shift	ARL		IRL					
Arithmetic left shift	ALS	LLS	LLS					
Arithmetic right shift	ARS	IRS	IRS					
Rotate left shift	ALR		LLR					

Table 6-21 (continued)
Datatypes Summary and Instructions in S, R, and V Modes

Operation	Size of Datatype (in Bits)							
	16 (A)	31 (A/B)	32 (L)	64 (L/E)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Rotate right shift	ARR		LRR					
Clear	CRA	CRL	CRL	CRLE				
Clear left	CAL	CRA	CRA	CRL				
Clear right	CAR	CRB	CRB	CRE				
Interchange halves	ICA	IAB	IAB	ILE				
Interchange and clear left	ICL	XCA	XCA					
Interchange and clear right	ICR	XCB	XCB					
Two's complement	TCA		TCL		FCM	DFCM	QFCM	
Set sign	SSM	SSM	SSM					
Clear sign	SSP	SSP	SSP					
Change sign	CHS		CHS					
Convert datatypes:								
Integer to floating point	FLTA	FLOT	FLTL					
Floating point to integer	INTA	INT	INTL				QINQ QIQR	
Binary to decimal	XBTD		XBTD	XBTD				
Decimal to binary	XDTB		XDTB	XDTB				
Position for integer divide	PIDA	PID	PIDL	PIDL				
Position after multiply	PIMA	PIM	PIML	PIML				
Skips	Saa				FSaa	FSaa		

Table 6-22
Datatypes Summary and Instructions in I Mode

Operation	Size of Datatype (in Bits)						
	16 (r)	32 (R)	64 (R/R+1)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Load from memory	LH	L		FL	DFL	QFLD	XMV
Store to memory	STH	ST		FST	DFST	QFST	
Add	AH	A		FA	DFA	QFAD	XAD
Subtract	SH	S		FS	DFS	QFSB	XAD
Multiply	MH	M		FM	DFM	QFMP	XMP
Divide	DH	D		FDV	DFDV	QFDV	XDV
Increment	IMH, IH1, IH2	IM, IR1, IR2					
Decrement	DMH, DH1, DH2	DM, DR1, DR2					
AND	NH	N					
OR	OH	O					
XOR	XH	X					
Complement	CMH	CMR					
Compare	CH	C		FC	DFC	QFC	XCM
Logical test	LHaa	Laa		LFaa	LFaa		
Branch	BHaa	BRaa		BFaa	BFaa		
Logical shift		SHL					
Arithmetic shift		SHA					
Shift right 1	SHR1	SR1					
Shift right 2	SHR2	SR2					
Shift left 1	SHL1	SL1					
	LHL1						

Table 6-22 (continued)
Datatypes Summary and Instructions in I Mode

Operation	Size of Datatype (in Bits)						
	16 (r)	32 (R)	64 (R/R+1)	32FP (FAC)	64FP (DAC)	128FP (QAC)	Dec (-)
Shift left 2	SHL2 LHL2	SL2					
Shift left 3	LHL3						
Rotate		ROT					
Clear		CR					
Clear left	CRBL	CRHL					
Clear right	CRBR	CRHR					
Interchange halves	IRB	IRH	I				
Interchange and clear left	ICBL	ICHL					
Interchange and clear right	ICBR	ICHR					
Two's complement	TCH	TC		FCM	DFCM	QFCM	
Set sign	SSM	SSM					
Clear sign	SSP	SSP					
Change sign	CHS	CHS					
Convert datatypes: Integer to floating point	FLTH	FLT					
Floating point to integer	INTH	INT				QINQ QIQR	
Binary to decimal	XBTD	XBTD	XBTD (DACO)				
Decimal to binary	XDTB	XDTB	XDTB (DACO)				
Position for integer divide	PIDH	PID	PID				
Position after multiply	PIMH	PIM	PIM				

SUMMARY

This chapter has introduced the datatypes supported on the 50 Series processors and has listed the instructions you can use to manipulate them. The next chapter, Altering Sequential Flow, lists instructions that allow you to test for a condition and perform actions depending on the outcome of the test.

7

Altering Sequential Flow

So far this document has confined its discussions mostly to arithmetic operations. This chapter describes instructions that can alter the normally sequential flow of control within a program.

BRANCH AND SKIP INSTRUCTIONS

The simplest way to change the flow of control in a program is to use a branch or a skip instruction. These instructions may directly load a new value into the program counter, or they may first test some value and then load the program counter according to the outcome of the test. Note that branch and skip instructions always load the program counter with an address contained within the current segment. (To transfer control to an address outside the current segment, use a jump instruction, explained in the second half of this chapter.)

Table 7-1 lists the branch instructions. Table 7-2 lists the logic test instructions. Table 7-3 contains information about the conditional skip instructions. Table 7-4 describes the floating-point skip instructions.

Table 7-1
Branch Instructions

Mnem	Name	Modes	Description
BEQ, BGE, BGT, BLE, BLT, BNE	Branch on A Set With Respect to 0	V	Branches if the contents of A meet the specified condition with respect to 0.
BCEQ, BOGE, BOGT, BCLE, BCLT, BCNE	Branch on OC Set With Respect to 0	V,I	Branches if the condition code reflects the specified condition with respect to 0.
BFEQ, BFGE, BFGT, BFLE, BFLT, BFNE	Branch on FA With Respect to 0	V,I	Branches if the contents of the floating accumulator reflect the specified condition with respect to 0.
BHEQ, BHGE, BHGT, BHLE, BHLT, BHNE	Branch on r With Respect to 0	I	Branches if the contents of the specified r meet the specified condition with respect to 0.
BLEQ, BLGE, BLGT, BLLE, BLLT, BLNE	Branch on L With Respect to 0	V	Branches if the contents of L meet the specified condition with respect to 0.
BMEQ, BMGE, BMGT, BMLE, BMLT, BMNE	Branch on Magnitude Condition Set With Respect to 0	V,I	Branches if LINK and the condition codes meet the the specified condition with respect to 0.
BREQ, BRGE, BRGT, BRLE, BRLT, BRNE	Branch on R Set With Respect to 0	I	Branches if the contents of the specified R meet the specified condition with respect to 0.
BRER	Branch on R Bit Reset	I	Branches if the specified bit in R is 0.
BRBS	Branch on R Bit Set	I	Branches if the specified bit in R is 1.
BHD1, BHD2, BHD4	Branch on r Decrementd by Value	I	Decrements r by the specified value and branches if the value is not equal to 0.
BHI1, BHI2, BHI4	Branch on r Incremented by Value	I	Increments r by the specified value and branches if the values is not equal to 0.

Table 7-1 (continued)
Branch Instructions

Mnem	Name	Modes	Description
BRD1, BRD2, BRD4	Branch on R Decrementd by Value	I	Decrements R by the specified value and branches if the value is not equal to 0.
BRI1, BRI2, BRI4	Branch on R Incremented by Value	I	Increments R by the specified value and branches if the values is not equal to 0.
BCS	Branch if CBIT is Set	V,I	Branches if the value of CBIT is 1.
BCR	Branch if CBIT is Reset	V,I	Branches if the value of CBIT is 0.
BLS	Branch if LINK is Set	V,I	Branches if the value of LINK is 1.
BLR	Branch if LINK is Reset	V,I	Branches if the value of LINK is 0.
BDX	Branch on Decrementd X	V	Decrements the contents of X by 1 and branches if the decremented value equals 0.
BDY	Branch on Decrementd Y	V	Decrements the contents of Y by 1 and branches if the decremented value equals 0.
BIX	Branch on Incremented X	V	Increments the contents of X by 1 and branches if the incremented value equals 0.
BIY	Branch on Incremented Y	V	Increments the contents of Y by 1 and branches if the incremented value equals 0.
CGT	Computed GOTO	V,I	Branches if the contents of A are greater than 1 and less than a specified integer; otherwise, executes the next instruction.

Table 7-2
Logic Test Instructions

Mnem	Name	Modes	Description
L EQ, L GE, L GT, L LE, L LT, L NE	Load on Register With Respect to 0	S,R,V,I	Loads a register with a 1 if the register reflects the specified condition with respect to 0; otherwise, clears the register to 0.
LCEQ, LGE, LOGT, LCLE, LCLT, LCNE	Load Register on Condition Codes Set With Respect to 0	S,R,V,I	Loads a register with a 1 if the condition codes reflect the specified condition with respect to 0; otherwise, clears the register to 0.
LFEQ, LFGE, LFGT, LFLE, LFLT, LFNE	Load Register on FAC With Respect to 0	S,R,V,I	Loads a register with a 1 if the contents of the floating accumulator reflect the specified condition with respect to 0; otherwise, clears the register to 0.
LHEQ, LHGE, LHGT, LHLE, LHLT, LHNE	Load R on r With Respect to 0	I	Loads R with a 1 if the contents of r reflect the specified condition with respect to 0, or with a 0 if another condition exists.
LLEQ, LIGE, LIGT, LILE, LILT, LINE	Load A on L With Respect to 0	S,R,V	Loads A with a 1 if the contents of L reflect the specified condition with respect to 0, or with a 0 if another condition exists.
LT LF	Load True Load False	S,R,V,I	Loads a register with a 1. Loads a register with a 0.

Table 7-3
Conditional Skip Instructions

Mnem	Name	Modes	Description
CAS	Compare A and Skip	S,R,V	Compares the contents of A to the contents of a memory location and skips depending on the outcome.
CAZ	Compare A to 0	S,R,V	Compares the contents of A to 0 and skips depending on the outcome.
CLS	Compare L and Skip	V	Compares the contents of L to the contents of a memory location and skips depending on the outcome.
DRX	Decrement and Replace X	S,R,V	Decrements the contents of X by 1 and skips the next 16 bits if the decremented value is 0.
IRS	Increment and Replace Memory	S,R,V	Increments the contents of a memory location and skips the next 16 bits if the incremented value is 0.
IRX	Increment and Replace X	S,R,V	Increments the contents of X and skips the next 16 bits if the incremented value is 0.
SAR	Skip on A Register Bit 0	S,R,V	Skips the next 16 bits if the specified bit in A contains 0.
SAS	Skip on A Register Bit 1	S,R,V	Skips the next 16 bits if the specified bit in A contains 1.
SGT	Skip on A Greater than 0	S,R,V	Skips the next 16 bits if the contents of A are greater than 0.
SLE	Skip on A Less Than 0	S,R,V	Skips the next 16 bits if the contents of A are less than 0.
SLN	Skip on LSB of A Nonzero	S,R,V	Skips the next 16 bits if bit 16 of A contains 1.
SLZ	Skip on LSB of A Zero	S,R,V	Skips the next 16 bits if bit 16 of A contains 0.
SMCR	Skip on Machine Check Zero	S,R,V	Skips the next 16 bits if the machine check flag contains 0.
SMCS	Skip on Machine Check Set to 1	S,R,V	Skips the next 16 bits if the machine check flag contains 1.
SMI	Skip on A Minus	S,R,V	Skips the next 16 bits if the contents of A are less than 0.
SNZ	Skip on A Nonzero		Skips the next 16 bits if the contents of A are not equal to 0.
SPL	Skip on A Plus	S,R,V	Skips the next 16 bits if the contents of A are greater than or equal to 0.
SRC	Skip on CBIT 0	S,R,V	Skips the next 16 bits if the value of CBIT is 0.
SSC	Skip on CBIT 1	S,R,V	Skips the next 16 bits if the value of CBIT is 1.
SZE	Skip on A Zero	S,R,V	Skips the next 16 bits if the contents of A are equal to 0.

Table 7-4
Floating-point Skip Instructions

Mnem	Name	Modes	Description
FSGT	Floating Skip If Greater Than 0	R,V	Skips the next location if the contents of the floating accumulator are greater than 0.
FSLE	Floating Skip If Less Than or Equal to 0	R,V	Skips the next location if the contents of the floating accumulator are less than or equal to 0.
FSMI	Floating Skip If Minus	R,V	Skips the next location if the contents of the floating accumulator are less than 0.
FSNZ	Floating Skip If Not Zero	R,V	Skips the next location if the contents of the floating accumulator are not equal to 0.
FSPL	Floating Skip If Plus	R,V	Skips the next location if the contents of the floating accumulator are greater than 0.
FSZE	Floating Skip If Zero	R,V	Skips the next location if the contents of the floating accumulator are equal to 0.

JUMP INSTRUCTIONS

Like the instructions listed in the tables above, jump instructions can load new addresses into the program counter. The difference is that jump instructions can transfer control to addresses outside the current segment of execution. Table 7-5 lists these instructions.

Table 7-5
Jump Instructions

Mnem	Name	Modes	Description
JDX	Jump on Decrement X	R	Decrements the contents of X by 1 and jumps if the decremented value is 0.
JIX	Jump on Increment X	R	Increments the contents of X by 1 and jumps if the incremented value is 0.
JMP	Unconditional Jump	S,R,V,I	Jumps to the specified effective address.
JSR	Jump to Subroutine	I	Jumps to the specified effective address and saves the return address in r.
JST	Jump and Store	S,R,V	Stores the current contents of the program counter into memory and jumps to the specified effective address.
JSX	Jump and Save in X	R,V	Increments the contents of the program counter by 1 and stores the result in X, then jumps to the specified effective address.
JSXB	Jump and Save in XB	V,I	Stores the current contents of the program counter in XB and jumps to the specified effective address.
JSY	Jump and Save in Y	V	Increments the contents of the program counter by 1 and stores the result in Y, then jumps to the specified effective address.

SUMMARY

The 50 Series supports branch, skip, and jump instructions that you can use to transfer control from one part of your program to another. The next chapter begins the discussion of more complex methods of control transfers.

8

Stacks and Procedure Calls

This chapter describes how to transfer control from one procedure to another. This type of control transfer, the procedure call, can:

- Call inward rings from outward rings.
- Invoke reentrant procedures.
- Invoke recursive procedures.
- Use an embedded operating system.

Before describing how procedure calls work, however, this chapter defines several key terms. It also describes the stack, the data blocks that contain information about a call, and the special access rights that govern a call.

DEFINITION OF TERMS

Process and Procedure

A procedure is a set of instructions, such as the body of a text editor or diagnostic program. A process is the execution of a procedure, such as the process that the system assigns to a user. A process may execute several procedures throughout its life.

A procedure may call other procedures by using the Procedure Call (PCL) instruction. A processor may exchange one process for another by invoking the process exchange mechanism (PXM). For information about the PXM, refer to Chapter 9 and Appendix C.

Note the use of the terms caller, callee, calling procedure, and called procedure. The procedure making the call is the calling procedure, or caller. The procedure answering the call is the called procedure, or callee. These terms are used throughout this and later chapters.

STACKS AND STACK MANAGEMENT

The more sophisticated methods of altering sequential program flow use stacks as temporary storage areas. Procedure calls use the stack to save the state of the machine before altering program flow and to contain the parameters of the call. When the specified operation is complete, information in the stack is used to restore the machine state to what it was before the procedure call took place.

Stacks

A stack is a group of one or more segments. Since a 50 Series processor can support more than one stack at a time, the segment number of the first segment in each stack (the stack root) serves as a unique identifier. Stack segments following the stack root segment are called stack extension segments. A stack can contain many stack extension segments.

Stack Header

The first four locations of the stack root segment contain the stack header. These locations contain information needed by the processor to manage the stack. Table 8-1 shows the format of these locations.

Each stack extension segment also has a header. Offsets 0 to 1 of each extension segment must contain 0. Offsets 2 to 3 contain an extension pointer that references the next stack extension segment. This pointer contains 0 if this segment is the last stack extension segment.

Table 8-1
Stack Header Format for the Initial Stack Segment

Offset	Name	Description
0, 1	Free Pointer	Pointer to first offset of next free space in the current stack segment (segment number/offset number). This value must be even.
2, 3	Stack Extension Pointer	Pointer to first location of extension segment, if one has been allocated. If there is not enough room to allocate a new frame in the current segment referenced by the free pointer, the processor uses the extension pointer to reference the next segment. If the extension pointer contains 0, no extension segment has been allocated and a stack overflow fault occurs.

Stack Frames

The 50 Series processors store information on the stack in blocks called stack frames. They allocate the frames in a last in first out (LIFO) manner. Each time the PCL instruction executes, a new frame is allocated; a PRTN instruction deallocates the frame when the procedure specified by PCL completes execution. An unextended frame cannot cross a segment boundary. (See STEX in the Instruction Sets Guide.)

The stack frames allocated at any time are backward threaded only. This means that each frame points back to the frame of the procedure that previously used this stack.

The information contained in a frame header defines the state of the machine that was in effect when the calling procedure executed the PCL instruction. This arrangement permits calls to or returns from a procedure without having to reference the frame of the calling procedure.

Figure 8-1 shows the format of the stack frame header. All procedures in the same ring can use the same stack for storage. Different processes, however, usually do not share stack segments.

FLAG BITS	0
STACK ROOT SEGMENT #	1
RETURN POINTER	2
RETURN POINTER	3
STACK BASE	4
STACK BASE	5
LINK BASE	6
LINK BASE	7
KEYS	8
ARGUMENT OFFSET #	9

Offsets In Frame	Contents	Description
0	Flag Bits	PCL always sets these bits to 0.
1	Stack Root Segment #	Address of the free pointer.
2 to 3	Return Pointer	Pointer to return location (that following the last argument template of the PCL instruction that created this frame).
4 to 5	Stack Base	Contents of caller's SB (pointer to previous frame).
6 to 7	Link Base	Contents of caller's LB.
8	Keys	Contents of caller's keys.
9	Argument Offset #	Offset number of the location following the PCL that created this frame.

Stack Frame Format
Figure 8-1

ENTRY CONTROL BLOCKS

The entry control block (ECB) identifies a procedure. When PCL executes, it forms the effective address of the called procedure's ECB, not of the procedure itself. The ECB contains information about the called procedure, as well as about the expected parameters (such as number of expected arguments, size of stack frame, and so on). Figure 8-2 shows the format and contents of the ECB.

1	16	17	32
ECB.PBH	ECB.PEL		
ECB.SFSIZE	ECB.ROOTSN		
ECB.ARGDISP	ECB.NARGS		
ECB.LEH	ECB.LBL		
ECB.KEYSH	0		
0	0		
0	0		
0	0		

Offset In Block	Name	Description
0 to 1	ECB.PB	Pointer (ring, segment, offset number) to the first executable instruction of the called procedure.
2	ECB.SFSIZE	Stack frame size to create (in half-words). Must be even.
3	ECB.ROOTSN	Stack root segment number. If zero, keep same stack.
4	ECB.ARGDISP	Displacement in new frame of where to build argument list.
5	ECB.NARGS	Number of arguments expected.
6 to 7	ECB.LB	Pointer (ring, segment, offset) to be loaded as called procedure's linkage base (location of called procedure's linkage frame less '400).
8	ECB.KEYS	Keys desired by called procedure.
9 to 15		Reserved, must be zero.

Entry Control Block Format
Figure 8-2

INDIRECT POINTERS

If the callee expects arguments, several pointers to the arguments should follow the PCL instruction. These pointers are called argument templates (or argument pointers), and contain orders which PCL uses to form indirect pointers to the actual arguments. Indirect pointers are saved in a stack frame that the callee uses to reference the arguments.

Several templates may be used in succession to form one indirect pointer. One template may specify a level of indirection; the next, a base register. Each template contains an S bit that determines if that template is the last one to be used to form a single indirect pointer. If this S bit contains a 1, then the argument is the last one to be used for this indirect pointer, and the processor should store it into the current stack frame. If the S bit contains a 0, then the indirect pointer requires more templates.

Each template also contains an L bit to indicate if it is the last one for the last indirect pointer. When L and S are both 1, then this argument is the last one for the last pointer. When L is 0, other arguments follow it. When L is 1 and S is 0, the processor stores the results of the current AP (argument pointer) into XB and (if necessary) in X. (See Storing Indirect Pointers, below, for information about these pointers.) In all cases, when the L bit is set to 1, no further APs are processed, and control is transferred to the called procedure. Figure 8-3 shows the format of all argument templates. Figure 3-3 in Chapter 3 shows the format of 32-bit and 48-bit indirect pointers.

1	4	5	6	7	8	9	10	11	16	17	32
BIT	I	0	ER	L	S	000000	OFFSET				

Bits	Mnem	Contents
1 to 4	BIT	Bit number
5	I	Indirect
6	---	Reserved; must be 0
7 to 8	ER	Base register
9	L	Last template for this call
10	S	Last template for this argument.
		If 1, store argument address to memory.
		If 0, store argument address to XB and X.
11 to 16	---	Reserved; must be zero
17 to 32	OFFSET	Offset number

Argument Template Format
Figure 8-3

GATE ACCESS

There are some Ring 0 or Ring 1 procedures that procedures in higher-numbered rings will want to call. Since normal read, write, and execute access rights will not allow such inward references, these Ring 0 or Ring 1 procedures must specify a special access right called gate access. Gate access allows a Ring 3 procedure to safely use a specific set of Ring 0 and Ring 1 procedures without harming the rest of the system.

For identification, the ECBs of the procedures that allow gate accesses are grouped in a special gate access segment. These ECBs must all have starting addresses of 0(mod16) in this segment. If a procedure references an improperly aligned ECB, an access fault occurs.

To call any of the procedures allowing gate accesses, the caller must execute a PCL instruction that points to an ECB in the gate access segment. There is no other way to call these procedures.

MAKING A PROCEDURE CALL

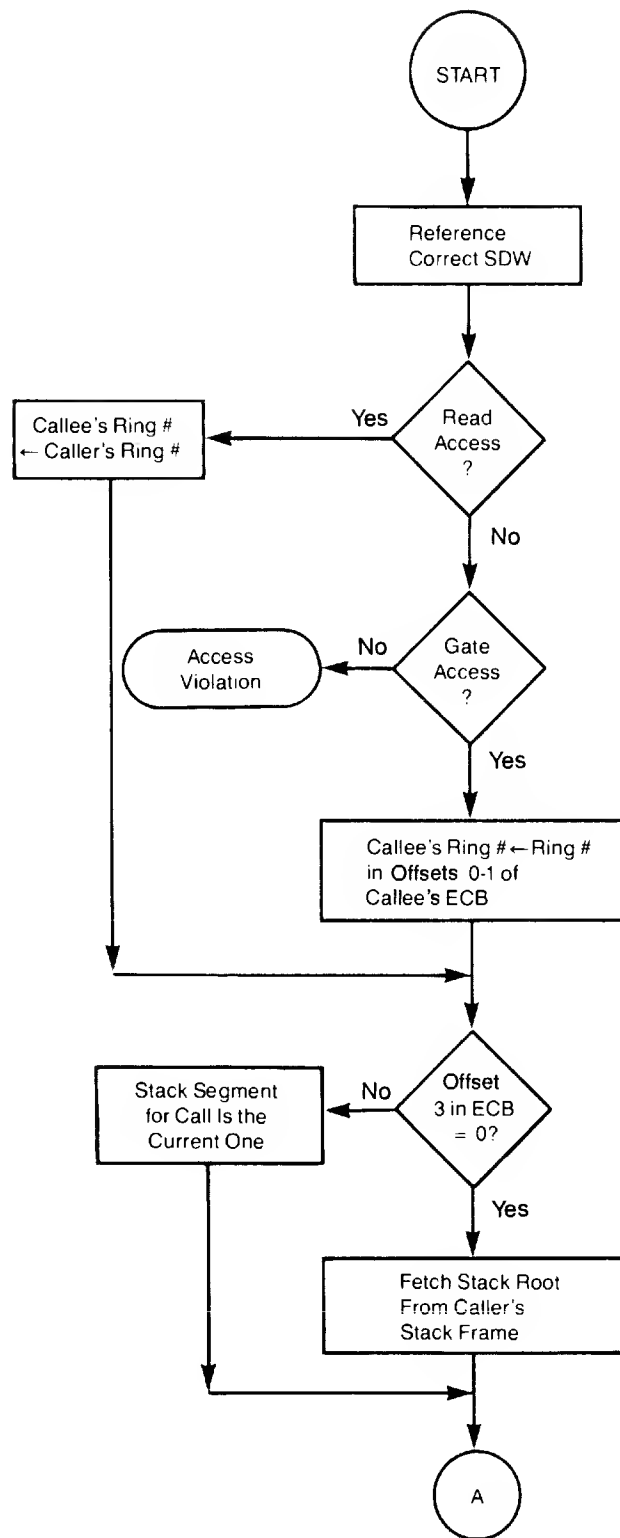
When PCL executes, it:

- Calculates the callee's ring number.
- Allocates a new stack frame for the callee.
- Saves the caller's state.
- Loads the callee's state.
- Calculates and stores indirect pointers for the callee's use.

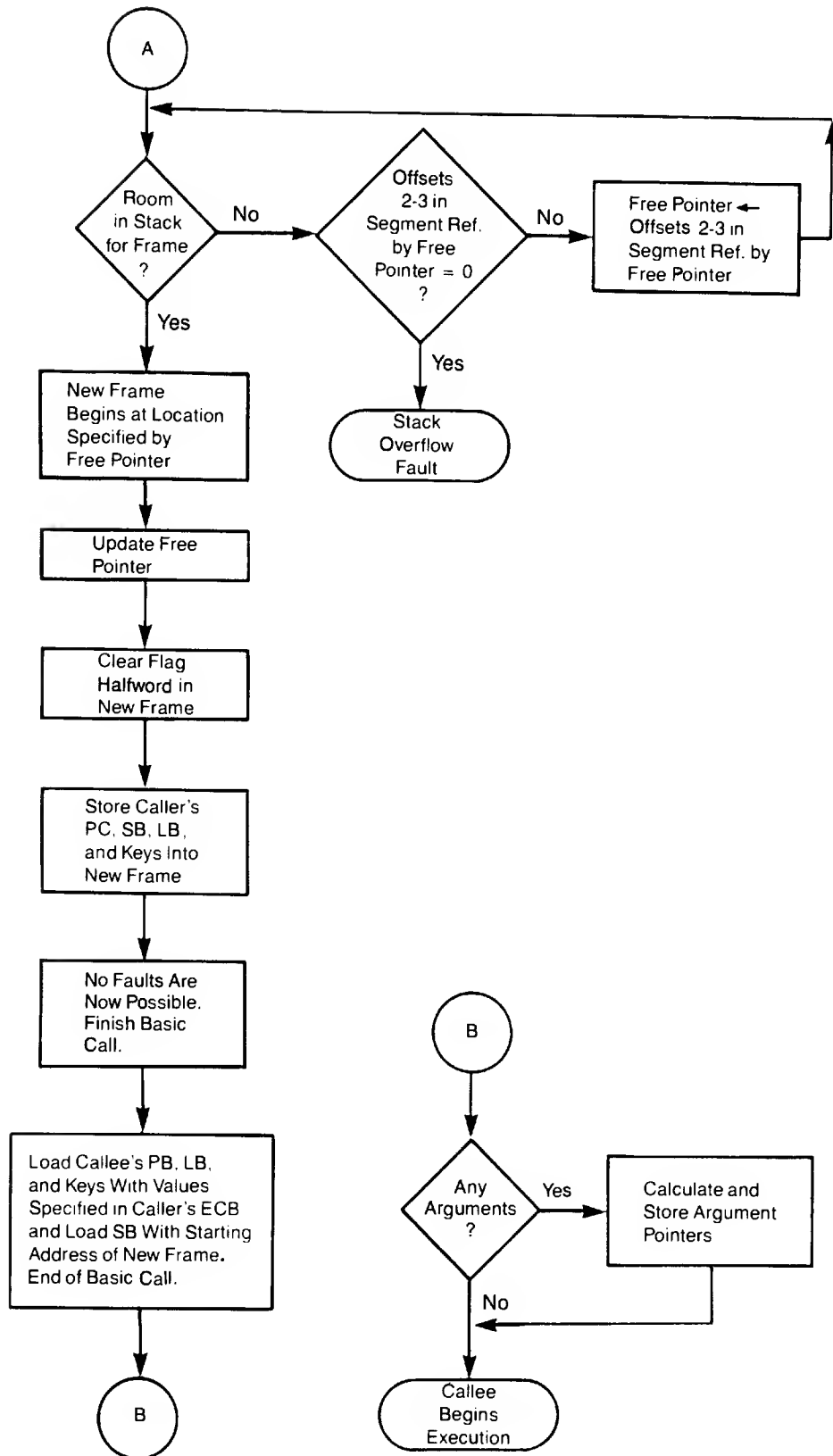
This sequence of events is summarized in Figure 8-4 and described below.

Calculating a Ring Number

When PCL begins execution, it calculates the ring number of the call. PCL looks at the appropriate STLB entry, since it contains access rights for the calling procedure. PCL uses these access rights to determine if the caller has access to the callee's ECB. If the STLB specifies read access, PCL weakens the ring number contained in the callee's ring field to that of the caller. If the callee's ECB is in a gate segment, PCL uses the ring field contained in offsets 0 to 1 of the callee's ECB as the ring number.



Actions of PCL, Part 1
Figure 8-4a



Actions of PCL, Part 2
Figure 8-4b

Allocating a Stack

PCL looks at the contents of ECB.ROOTSN (offset 3 of the ECB) to determine the stack root segment. If ECB.ROOTSN contains zeroes, the processor fetches the stack root number from the stack frame of the caller. (Gate ECBs must have a nonzero stack root segment indicated in ECB.ROOTSN.) The first two offsets of the stack root segment contain the free pointer; PCL compares the number of available locations in the segment to the contents of ECB.SFSIZE (the number of 16-bit quantities contained in a frame). Stack frame sizes and free pointers are always rounded upwards to form an even value.

If the frame will fit into the locations remaining in the stack segment, PCL starts the new frame at the location specified by the free pointer. It also updates the contents of the free pointer so that they point past the new frame.

If the new frame is too large to fit in the current segment, PCL examines the contents of offsets 2 to 3 in the segment referenced by the free pointer. If offsets 2 to 3 contain 0, a stack overflow fault occurs.

If offsets 2 to 3 contain a nonzero value, this value becomes the new free pointer. PCL rechecks for available segment locations as it did the first segment. If this segment cannot contain the whole frame, a stack fault occurs. If there are enough available locations, PCL starts the frame at the first available location.

Saving the Caller's State

The processor clears the flag field of the new frame and stores the contents of the caller's program counter, stack base and link base registers, and keys into the new frame. The contents of the saved program counter specify the ring and segment of the caller. These saved contents point to the location immediately following PCL.

Loading the Callee's State

At this point, no faults are possible and the basic call must be finished. PCL loads the program counter with the contents of ECB.PB and LB with the contents of ECB.LB. The keys are loaded with the contents of ECB.KEYS; note, however, that bits 15 to 16 of the keys are reset to 0. PCL also loads the address of the new frame into SB. This is the end of the basic call. If there are any arguments, PCL must calculate and store the argument pointers before beginning execution of the procedure.

Calculating Indirect Pointers

Figure 8-5 shows how the indirect pointers are formed. The text that follows elaborates on this figure.

To form an indirect pointer, PCL first forms the ring field. It compares the contents of the program counter's ring field and that of the base register specified in the caller. The larger value of these two fields becomes the ring field of the indirect pointer.

The contents of the segment field of the caller's specified base register become the contents of the indirect pointer's segment field.

The contents of the base register's offset is added to the offset field of the argument template. If the specified base register is not XB, the contents of the bit field of the argument template become the bit field of the indirect pointer. When the specified base register in the argument template is XB, the bit field of the template is added to the bit field of the argument pointer saved in the XB and X registers, and any carryout goes to the offset field of the indirect pointer. The bit field is ignored if the indirect bit contains a 1 in the argument template.

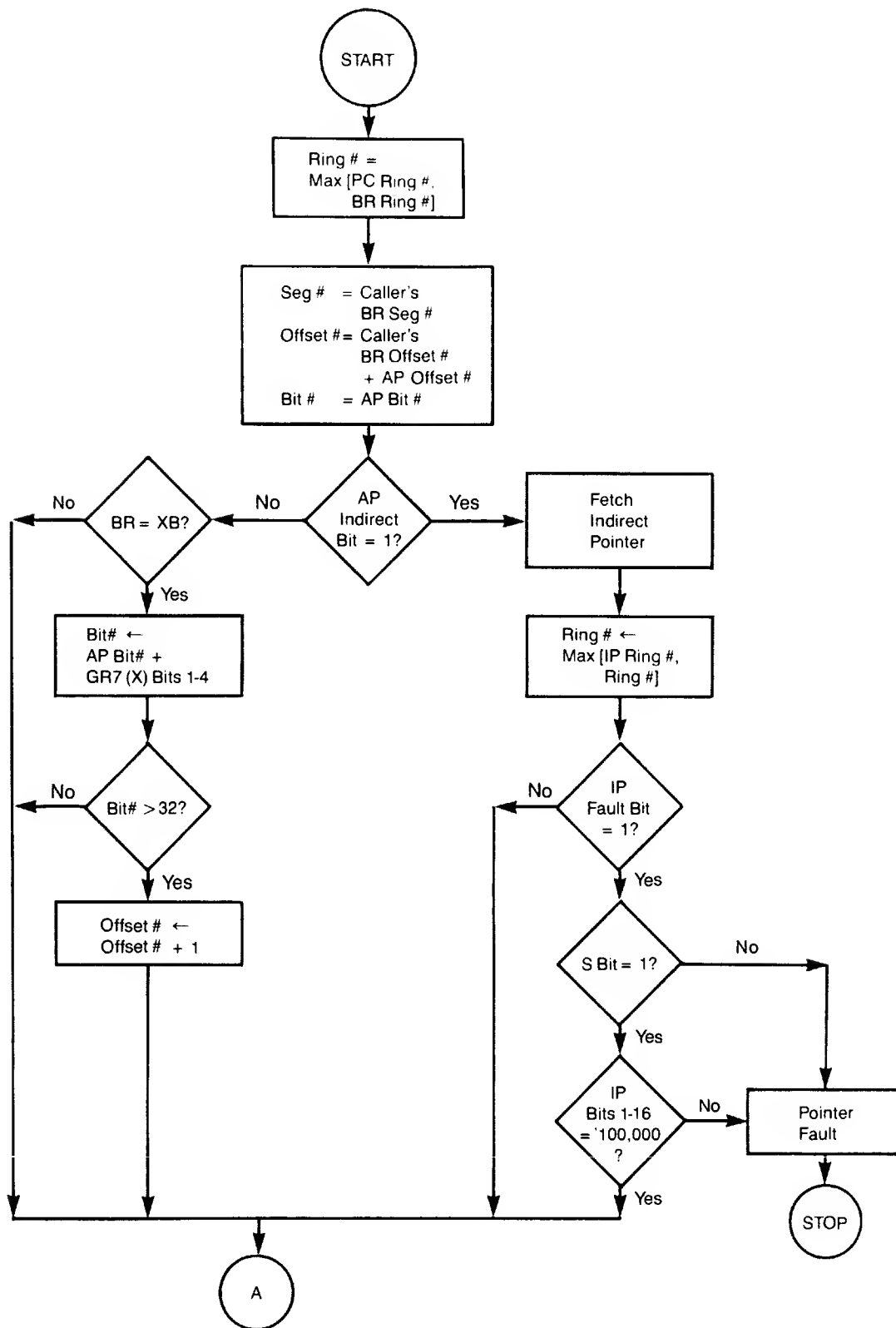
If the argument template indirect bit contains a 0, the value just calculated is the final value.

If the argument template indirect bit contains a 1, the value just calculated is not the final value. PCL uses this calculated value to fetch the indirect pointer. PCL compares the calculated value's ring field to the caller's ring field (found in the program counter) and takes the larger of the two as the new ring field. The contents of the segment, offset, and bit fields are the same as the contents of those in the fetched indirect pointer.

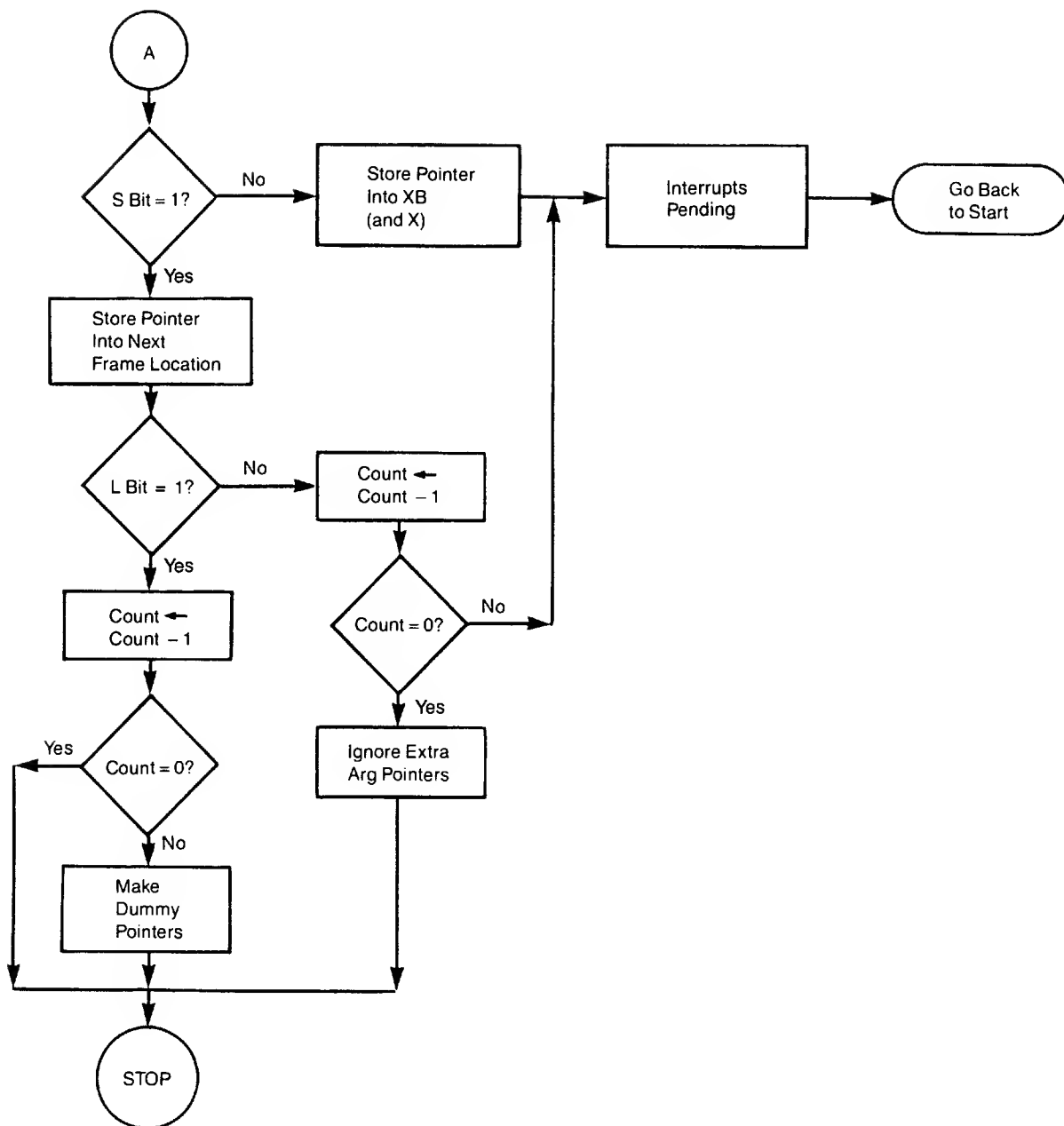
When an indirect pointer's fault bit contains a 1, the contents of the argument template S bit and the pointer's first 16 bits determine the action to be taken. If the S bit contains a 1 and the pointer's first 16 bits are '100000, the indirect pointer is loaded onto the callee's stack frame; all other cases result in a pointer fault.

Once PCL finds the final value generated by the template, it examines the S bit to determine if it should store the pointer in the stack frame as an indirect pointer, or if it should store the pointer in XB.

If S contains a 0, PCL must use at least one more template to complete the formation of the pointer. The value calculated so far is stored in XB. (If there is a bit field, the value is also stored in X. Bit 4 of XB, the E bit, contains 1 when X is used.) The value calculated for the next template is stored in XB and X again. XB is only updated whenever the S bit contains a 0. Otherwise, XB remains unchanged. This continues until the S bit or the L bit of one of the templates contains a 1.



Calculating and Storing Argument Pointers, Part 1
Figure 8-5a



Calculating and Storing Argument Pointers, Part 2
Figure 8-5b

Storing Indirect Pointers

If S contains a 1, PCL stores the calculated indirect pointer in the next stack frame location. If L also contains a 1, then there are no more indirect pointers to be calculated. A 0 in L indicates that there are more arguments to follow, so PCL proceeds with the next one.

If the number of indirect pointers produced is greater than the number the callee expects, PCL ignores the extras.

If the number of indirect pointers produced is less than the number the callee expects, PCL creates dummy indirect pointers and stores them in the current frame. The format of these dummy pointers is '100000, where bit 1 = 1 indicates a pointer fault (omitted argument pointer). PCL stores one dummy pointer for each omitted one.

The callee can reference omitted indirect pointers only to pass them on to other new procedures; if such a reference occurs, the new procedure will see such indirect pointers as omitted. Any use of an omitted indirect pointer other than to pass it on causes a pointer fault.

PCL always allocates three 16-bit quantities in the current stack frame to store each indirect pointer. An indirect pointer occupies all three 16-bit quantities, however, only if it has a nonzero bit field. If this is the case, PCL sets the E bit for that indirect pointer to 1. If an indirect pointer has a bit field containing 0, PCL sets the argument's E bit to 0 and loads the indirect pointer into the first two allocated locations; when PCL loads the next indirect pointer, it skips the third location.

THE ARGV INSTRUCTION

PCL is resumable if any interruption occurs while it is transferring arguments. When such an interruption occurs, the program counter in the return block contains the address of the first instruction in the callee. If the callee does not expect arguments, its first instruction can be anything. If arguments are expected, however, the first instruction of the callee must always be the Argument Transfer (ARGV) instruction. After the processor services the interrupt, control returns to ARGV, which identifies how many indirect pointers have yet to be transferred, and begins the transfer anew at that point.

ARGV transfers arguments only if an interrupt occurs during PCL's execution. If this happens, ARGV completes the transfer that PCL began. If no interrupt occurs, ARGV is not executed.

THE PRTN INSTRUCTION

After all arguments are transferred, control transfers to the called procedure. The last instruction of the called procedure must be a procedure return instruction, PRTN. When this called procedure completes execution, PRTN transfers control back to the calling procedure. The calling procedure picks up execution at the instruction immediately following PCL and its arguments.

PRTN also deallocates the stack frame created when the procedure call was first made. To deallocate the frame, the instruction stores the current value of the stack base register into the free pointer. It then restores the caller's state by loading the caller's stack base and link base registers with the values contained in the frame being deallocated. The keys are similarly loaded, but bits 15 to 16 of the keys are reset to 0. PRTN also loads the program counter with the appropriate address contained in the frame, but loads the program counter's ring field with the logical OR (weaker) of the saved program counter ring number and the current ring number. This prevents inward returns, yet allows returns from gated calls to work properly.

PROGRAMMING NOTES

When making a procedure call, make sure that the caller, callee, and associated ECB all contain consistent information about arguments. If the ECB specifies no arguments, then no argument templates should follow PCL, nor should the callee begin with ARGV. Similarly, if the ECB specifies arguments, the associated callee must begin with ARGV, and PCL should be followed by the correct number of argument templates (or fewer).

Also, PCL without argument pointers does not change the contents of any general registers or XB. PCL with argument pointers may alter the contents of some general registers, so do not rely on them to be the same as they were before PCL executed. Specifically, when calling an inner-ring procedure, do not use an indexed or an XB-relative PCL instruction. If an asynchronous interrupt condition occurs, the software restarts the interrupted call at the location specified by the calling PCL. Since neither XB nor the general registers were saved during the first try of PCL, the processor may calculate an invalid effective address.

In addition, do not specify an XB-relative argument template unless it is immediately preceded by at least one other template whose S bit contains a 0. The previous template's S bit tells the processor that another template is to follow, and to save the current template in XB, not to store it in memory. The processor reads in the XB-relative template, and uses the saved contents of XB in the manipulation. If the XB-relative template were not immediately preceded by another template whose S bit contains a 0 and if the processor were to retry PCL, XB would not contain valid contents; the calculated template would be invalid.

9

Process Exchange

INTRODUCTION

You read in the previous chapter how to transfer control from one procedure to another. This chapter and the next discuss the Process Exchange Mechanism (PXM) and how it transfers control from one process to another. This chapter describes the PXM implemented on all single-stream processors. Appendix C describes the PXM implemented on a dual-stream processor, the 850.

As defined in the previous chapter, a process is a dynamic state of execution, such as a user in a time-sharing system. To quickly service as many processes as possible (up to approximately 1000 at once), the 50 Series PXM executes one process for a given length of time. If a resource is not available or time for this process is up, the PXM exchanges this process for another, and so on. This allows many processes to work towards completion at the same time.

ELEMENTS OF THE PXM

The main elements of the process exchange mechanism (PXM) are:

- Three data structures:

- Process control blocks
- Ready list
- Wait lists

- Two PXM instructions:

WAIT
NOTIFY

- The dispatcher

In addition to these elements, the PXM manipulates the register file and the process interval timer during process exchange.

PROCESS CONTROL BLOCKS

Each process has a process control block (PCB) that describes it. Each PCB contains a minimum of 64 halfwords and completely specifies its process from a hardware point of view. Table 9-1 shows the format of the PCB.

A single segment contains the PCBs of all processes running throughout the system. Bits 1 to 16 of register '25 in the current register set specify the number of this segment, OWNERH. (See Table 9-5 later in this chapter for the format of the current register set.) The pointers and addresses in a PCB (except fault vectors and wait list pointers) are 16 bits long and are assumed to be relative to OWNERH. Note that for the 6350, the contents of the concealed stack can go anywhere in segment OWNERH + 1; for the rest of the 50 Series, those contents can go anywhere in OWNERH. (For more information on OWNERH, see the section on User Register Files, later in this chapter.)

PCBs generally start on 0(mod64) boundaries, but must start on at least 0(mod32) boundaries.

READY LIST

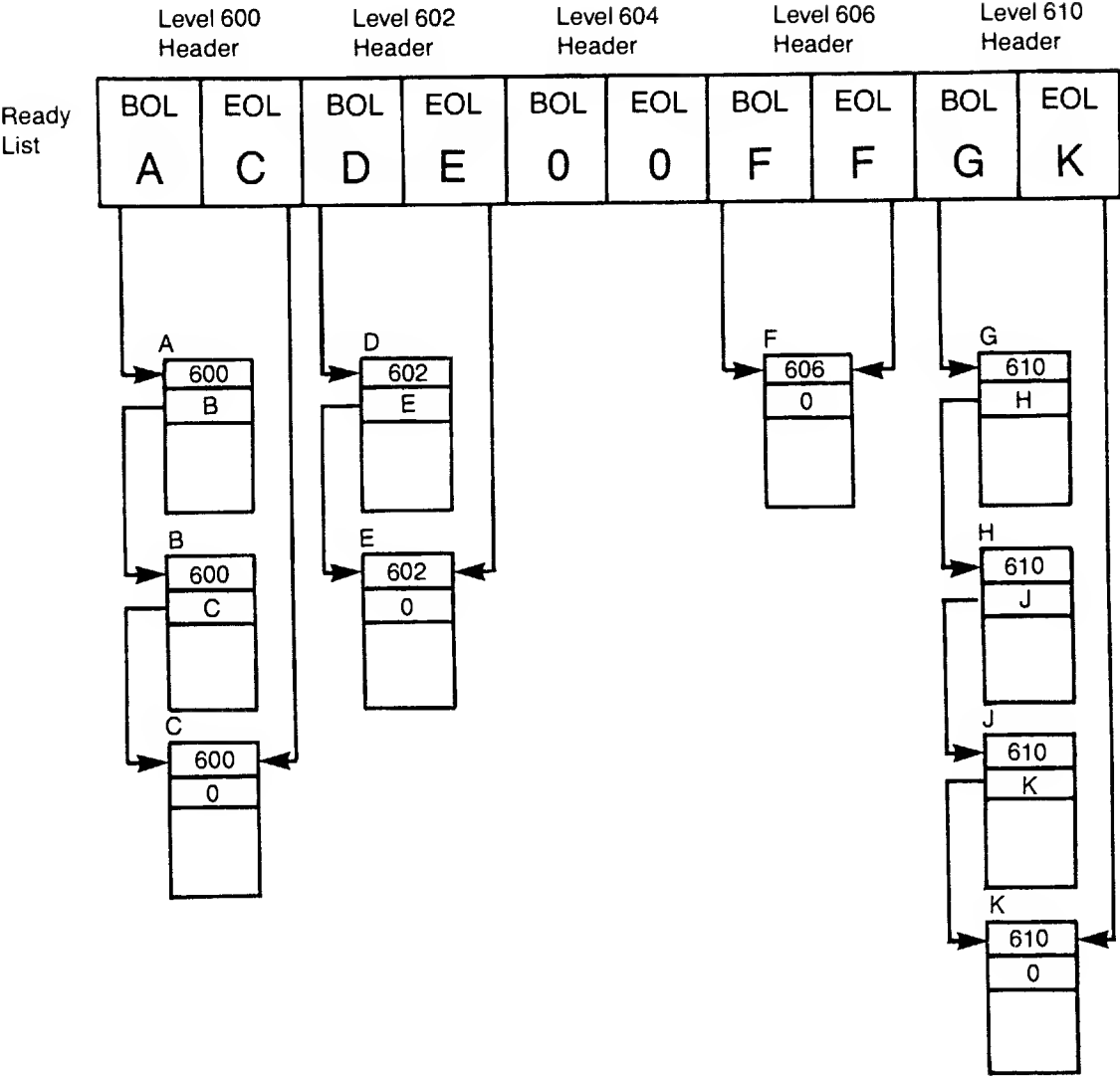
The PXM uses the ready list to indicate priorities and dispatch processes. The elements of the ready list are:

- A series of headers that make up the actual ready list
- A data base made up of PCBs
- Two 32-bit registers, PPA and PPB

Figure 9-1 and the text in the following section show the relationships between the ready list elements.

Table 9-1
PCB Format

Section	Offset	Contents
Control	0	Level pointer to BOL in ready list.
	1	Link pointer to next PCB, or 0.
	2 to 3	Segment number/offset number of the semaphore whose wait list is currently pointing to this PCB. A segment number of 0 indicates that this PCB is on the ready list.
	4	Abort flags used to generate a process fault when this PCB is dispatched. Bits 1 to 15: set by the software Bit 16: process interval timer overflow
	5	Pointer to the register set that this process used last.
Process State	6 to 7	Reserved for future use.
	'10 to '11	Elapsed timer. Must be maintained by the software that resets the live interval timer. Discussed further in Table 9-11.
	'12 to '15	DTAR2 and DTAR3. These are never saved, only restored. See Figure 4-9 and Table 9-5.
	'16 to '17	Interval timer (copy of TIMER, shown in Table 9-5). Discussed further in Table 9-11.
	'20	Save mask. PXM uses this to avoid saving or restoring registers containing zeroes. The format of the save mask is: 1 to 8: GRO-GR7 (see Table 9-5) 9 to 12: FAC0-FAC1 (see Tables 6-9 and 9-5) 13 to 16: PB, SB, LB, XB (see Table 9-5 and Figure 3-1).
Fault	'21	Keys. (See Figures 5-3 and 5-4, Table 9-5.)
	'22 to '61	Storage for nonzero registers. (See Save mask, above.)
	'62 to '63	Fault vector. Segment number/offset number to fault table for Ring 0.
	'64 to '65	Fault vector. Segment number/offset number to fault table for Ring 1.
	'66 to '67	Reserved for future use.
	'70 to '71	Fault vector. Segment number/offset number to fault table for Ring 3.
	'72 to '73	Fault vector. Segment number/offset number to fault table for page fault.
	'74 to '76	Concealed fault stack header (FIRST, NEXT, and LAST pointers). (See Table 10-7.)
	'77	Reserved.
	See note to the right	Concealed stack, whose contents can go anywhere in segment OWNERH + 1 (6350 only) or OWNERH (rest of 50 series) and can contain as many frames as desired. See the <u>Concealed Stack</u> section in Chapter 10.



Ready List and Associated PCB Lists
Figure 9-1

Headers

The ready list itself is made up of headers, one header for each level of priority. These headers are allocated in contiguous memory locations, with the highest priority header contained in the lowest numbered memory location. Each header, in turn, is made up of two 16-bit pointers. The pointers are called the beginning of list (BOL) pointer and the end of list (EOL) pointer, and each contains the address of a PCB in segment OWNERH.

The PCB referenced by a BOL pointer is associated with the first process having a particular priority. The EOL pointer points to the PCB of the last process with that particular priority.

A BOL pointer containing a 1 signals the end of the ready list, since PCB addresses must be even. A BOL pointer containing a 0 signals an empty level.

Ready List Data Base

The ready list data base is made up of linked lists of PCBs whose associated processes are ready to execute. There is one list defined for each level of priority; all PCBs contained in that list have the same level of priority. A list can contain as many processes as can exist in the system at a time.

The first location in each PCB specifies the process' priority level by pointing to one of the BOL pointers in the ready list. The second location contains a forward link to the next PCB in the linked list. For the last PCB in the linked list (that is, the last PCB in the ready list with this level of priority), the second location contains 0.

PPA and PPB Registers

The PKM uses the pointer to process A (PPA) and pointer to process B (PPB) registers to locate the next process to dispatch. Both registers are 32 bits wide.

PPA always contains information about the currently active process. Bits 17 to 32 contain PCBA, the address of the process' PCB. Bits 1 to 16 contain the level of priority, called Level A. Level A always specifies the system's highest priority level that has an associated PCB ready to run. This is because the system's currently running process is always the highest priority process that is capable of running.

PPB contains Level B and PCBB, which specify the priority level and the PCB address, respectively, of the next process to run when execution of the current process terminates.

Using PPA, PPB, and the Ready List

To show how PPA and PPB are used, suppose Process H is running when Process J, whose priority is higher than that of Process H, needs to be serviced. This means that Process J preempts Process H. The PXM suspends Process H, saves the contents of PPA (which reference Process H) in PPB, and then services Process J. When Process J completes, the PXM checks PPB to see what process to run next. PPB identifies Process H, and so the PXM resumes execution of Process H.

Except when bringing the system up from a cold start, software should never alter the contents of PPA or PPB. This holds even if PCBA or PCBB contains 0, indicating invalid register contents. Even if PCBA is invalid, Level A specifies the highest level of priority that was executing in the system, and this determines the starting point of a scan to find the next process to run. When PCBA is invalid, PCBB is guaranteed to be invalid. Note that PCBB is also invalid when the system is idle.

Upon cold start, the cold start software loads the PPA register with the highest level of priority in the ready list. At all other times, however, Level A specifies the highest level of priority that was last known to contain a process. All scans of the ready list can begin at this last known level. Whenever the PXM needs to run a process of higher priority than that specified in Level A, the PXM loads PPA with that higher level.

The PXM does not maintain a pointer to the highest priority level of the ready list. The ready list allocator that starts the PXM, however, knows the starting address of the ready list. In addition, Level A always points to either the highest priority level currently in the system, or the last known highest level. This means that Level A can be a pointer into the ready list.

If PCBB is valid, Level B points to the next process to be executed when the current process completes. The priority level of this next process is lower than or equal to that of the currently executing process. If PCBB is invalid, the contents of Level B are unpredictable.

WAIT LISTS

Wait lists specify a group of processes that are waiting for an event to occur. There are two major elements of each wait list:

- A semaphore
- A data base made up of PCBs

Figure 9-2 and the text in the following section describe the relationship between a semaphore and the wait list PCBs.

Semaphores

Semaphores define an event, such as the completion of a task. The definition of the semaphore is known by at least two processes, or by one process and phantom interrupt code. Upon completion of the event, a NOTIFY instruction changes the value of the semaphore. This change in value may cause the PXM to run a new process.

A semaphore consists of two sequential 16-bit memory locations. The first location contains a WAIT counter, C. If C is greater than zero, then it specifies the number of PCBs on the associated wait list. If C is negative, it specifies the number of times the event has occurred without running a process.

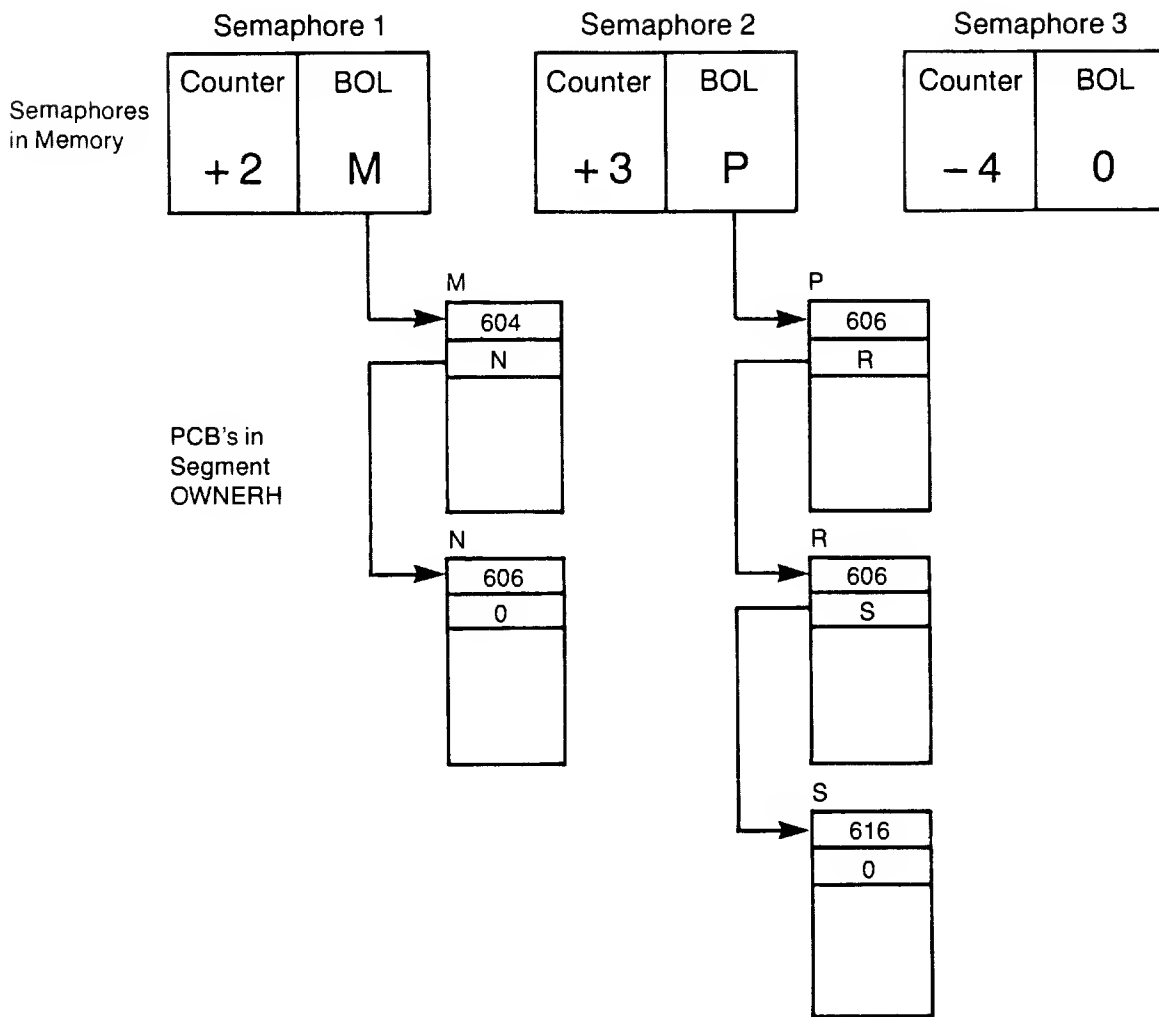
The second location contains the address of the first PCB awaiting completion of the specified event. Since all PCBs are contained in segment OWNERH, a 16-bit pointer is all that is needed to identify a specific PCB.

Semaphores can reside anywhere in memory but segment 0.

Wait List Data Base

Each wait list has associated with it a linked list of PCBs. The processes represented by the PCBs all share the same semaphore; this means that they are all waiting for the same event to occur.

The PCBs in a wait list need not have the same level of priority, since the wait list uses a priority-based queuing algorithm. This means that processes with higher priorities are queued ahead of those with lower priorities.



Wait List and Associated PCB Lists
Figure 9-2

PXM INSTRUCTIONS

The two notify instructions, NFYE and NFYB, and the wait instruction, WAIT, are restricted instructions. Therefore, they must be executed in Ring 0. All three instructions are 48 bits long: bits 1 to 16 contain an instruction code, and bits 17 to 48 contain a 32-bit address pointer to a semaphore.

The WAIT Instruction

Figures 9-3 and 9-4 show the actions of the WAIT instruction.

As the name indicates, WAIT signals the PXM to wait for an event before executing any more of the currently running process. When WAIT executes, the processor uses the address pointer contained in the instruction to reference a particular semaphore. The processor increments the counter contained in the addressed semaphore, then looks at the result.

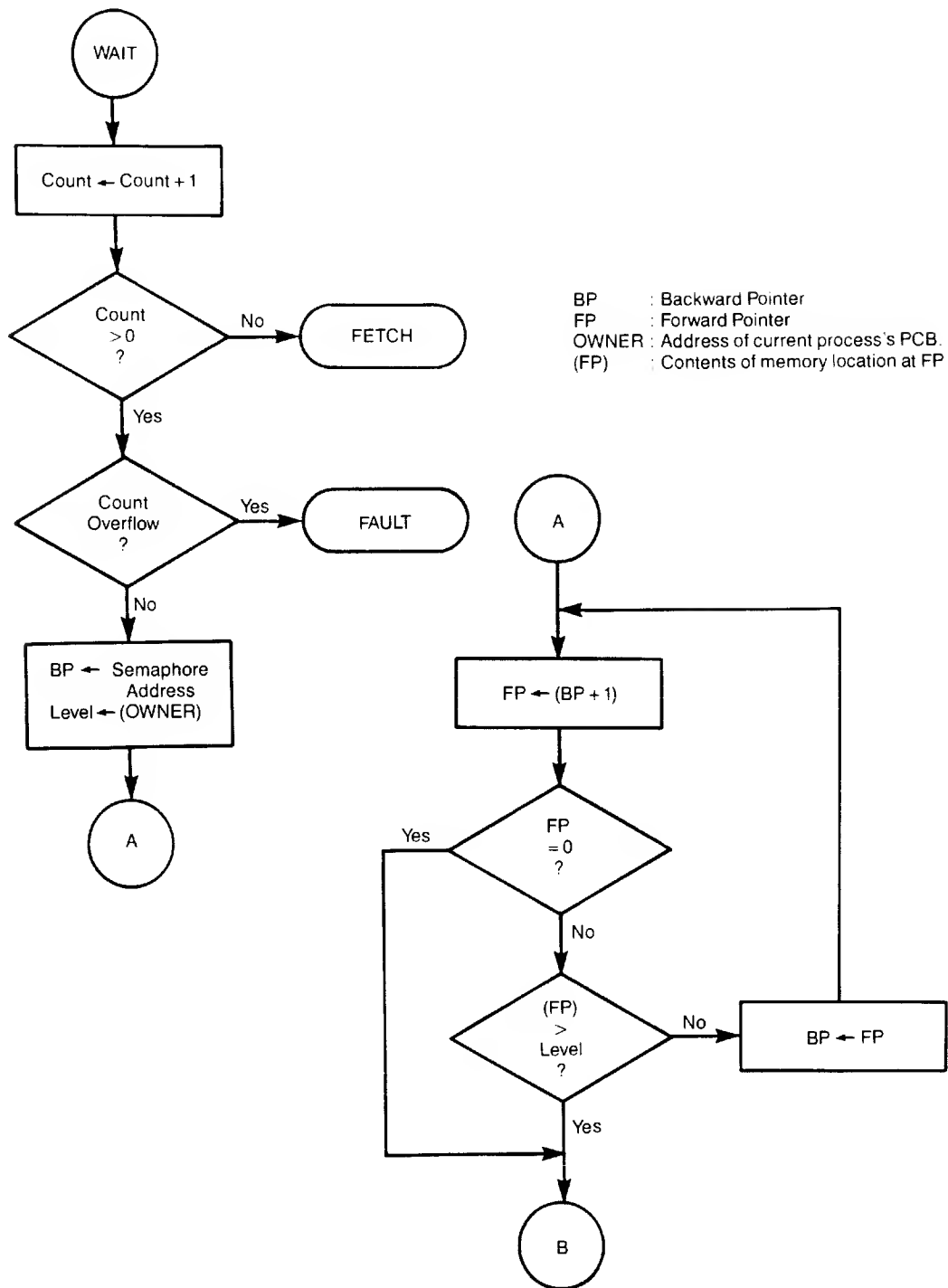
If the result is less than or equal to 0, there are no other processes waiting for the event defined by the semaphore. In this case, the currently executing process can continue.

If the result is greater than 0, either the expected result has not occurred, or the desired resource is not available. The processor stops executing the current process, removes the associated PCB from the ready list, and places the PCB on the wait list associated with the semaphore. The PCB's priority level dictates where on the wait list the PCB should go. If the wait list already contains PCBs with the same priority level, the new PCB is placed after the ones already there.

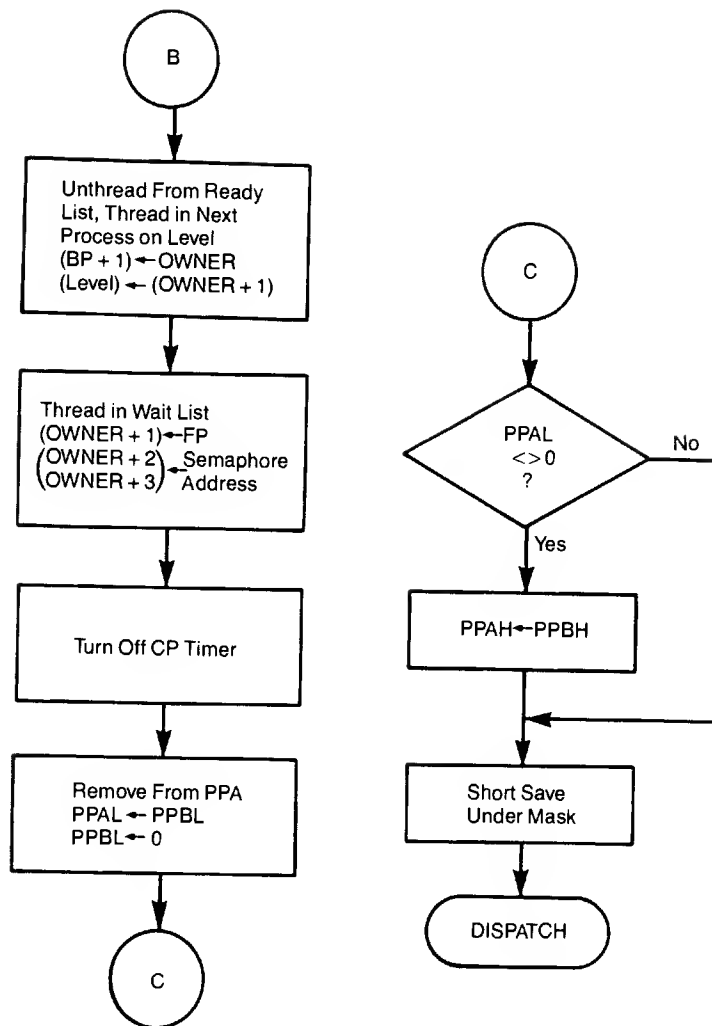
A semaphore overflow fault occurs if the result is greater than +32767. See Chapter 10 for details. This fault does not occur, however, for the earlier processors listed on page 1-1.

Note

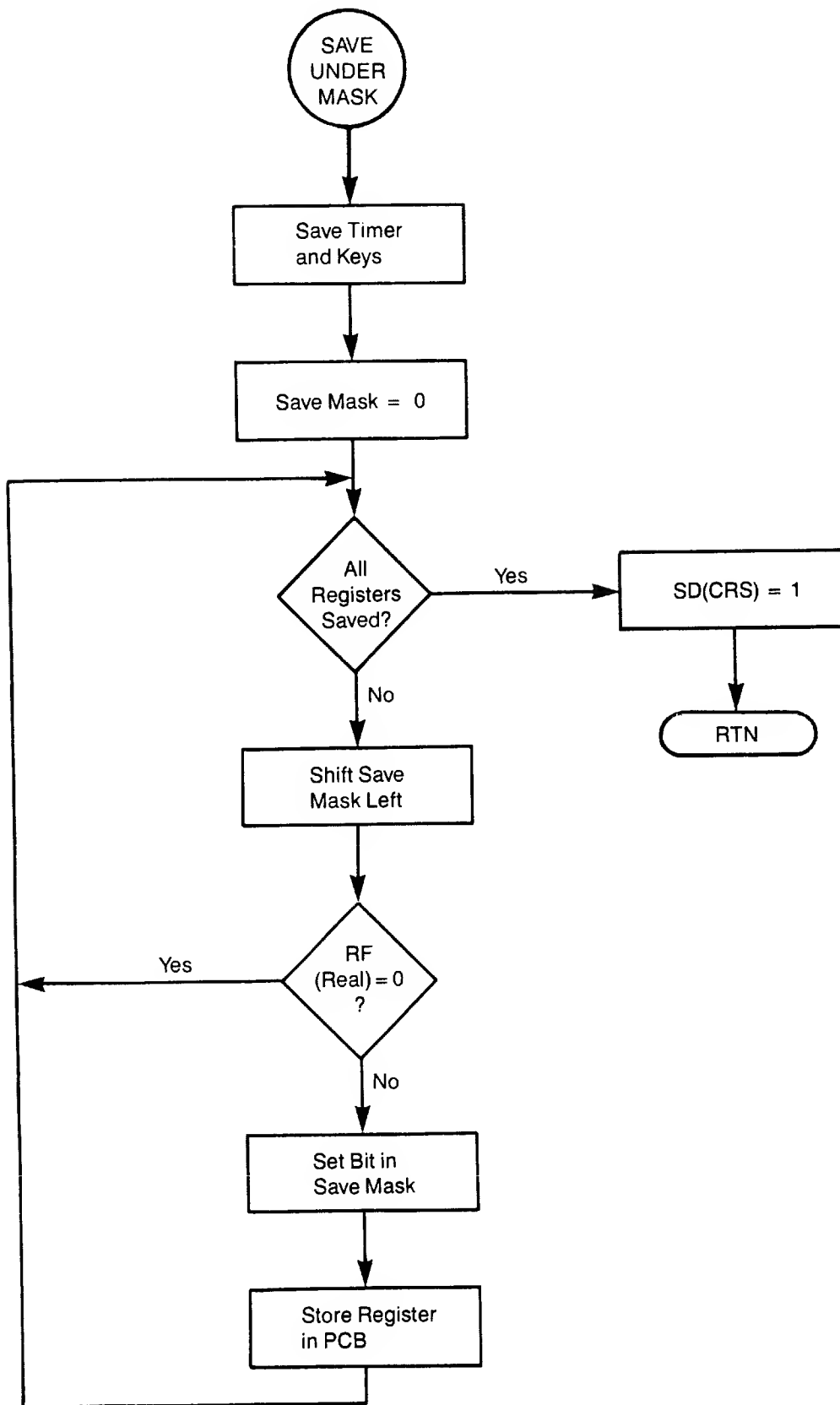
The processor saves only the contents of the keys, base registers, program counter, and the interval timer (down to the microsecond) when it adds a PCB to the wait list. It does not save the contents of the general registers or floating registers. After this short save the processor makes the register set used by the exchanged process available to the next process to run. For this reason, never assume that the contents of the general registers after a WAIT instruction executes are the same as they were before WAIT executed.



WAIT Instruction, Part 1
 Figure 9-3a



WAIT Instruction, Part 2
Figure 9-3b



Save Under Mask Algorithm
Figure 9-4

The NOTIFY Instructions

Figure 9-5 shows the actions of NOTIFY.

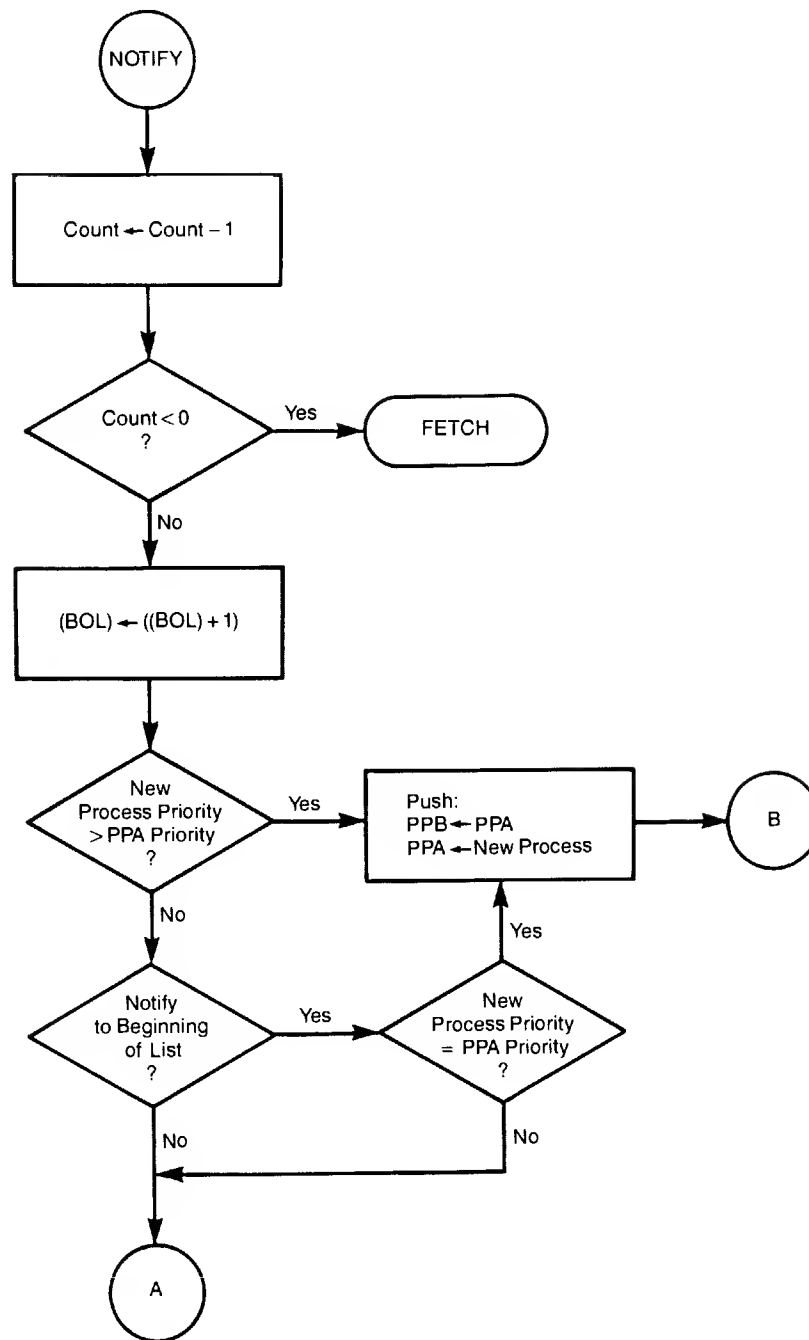
The two notify instructions, NFYE and NFYB, perform the same sequence of events. They differ only in the queuing algorithm used: NFYE queues PCBs at the end of the appropriate ready list priority level, while NFYB queues PCBs at the beginning of the appropriate priority level. In the discussion that follows, NOTIFY encompasses the operation of both instructions.

NOTIFY signals the PXM that some awaited event has occurred. When NOTIFY executes, the processor uses the address pointer contained in the instruction to reference a semaphore. The processor decrements the counter contained in the semaphore by 1 and checks the result.

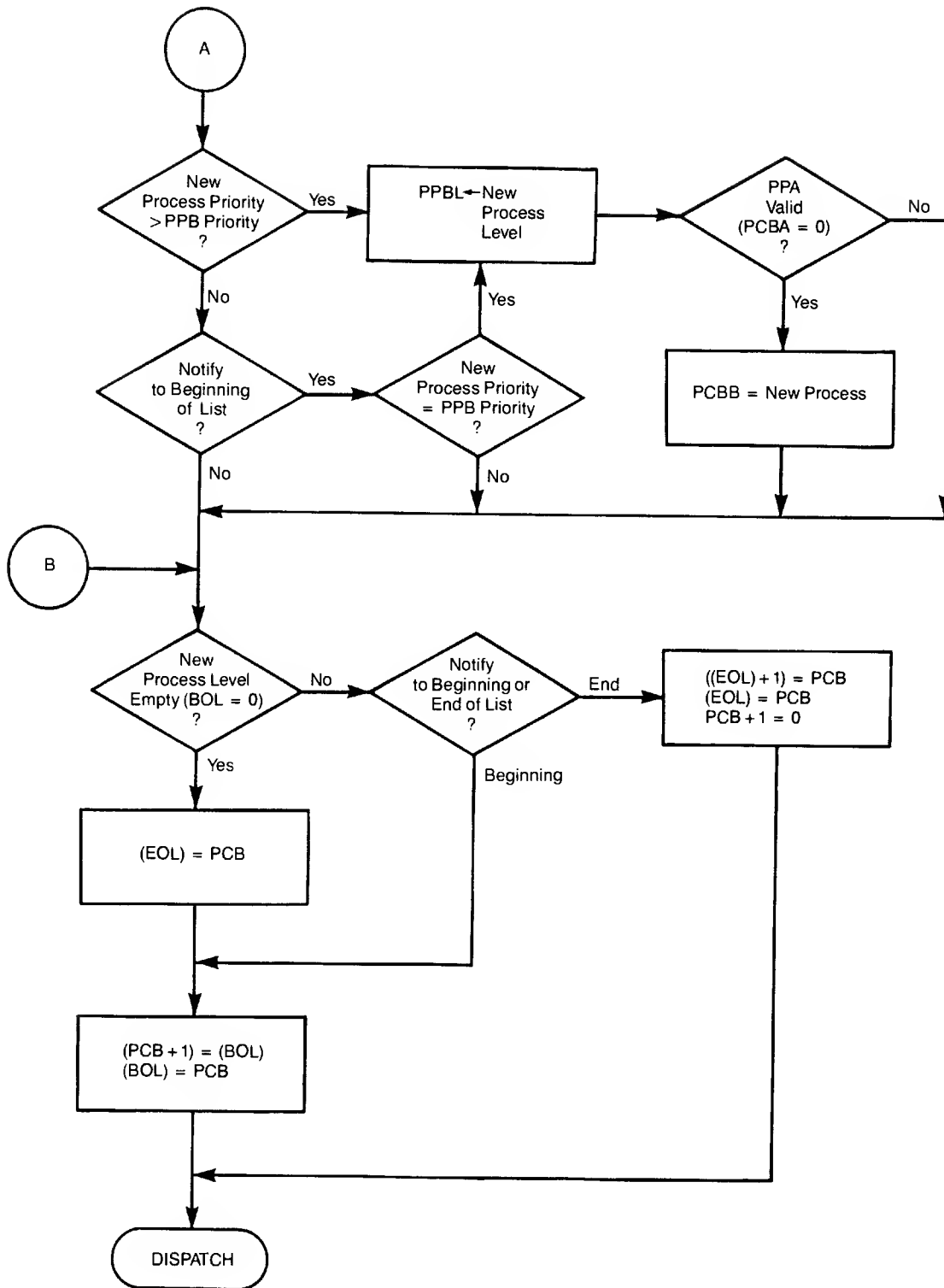
If the result is less than 0, no process is waiting for this event, so the processor continues the currently executing process. If the result is less than -32768, a semaphore underflow fault occurs. This fault, however, does not occur for the earlier processors listed on page 1-1.

If the result is greater than or equal to 0, the processor removes the PCB at the head of the specified wait list and places it on the ready list. If the process associated with the PCB moved to the ready list has a higher level of priority than that of the currently executing process, the processor will preempt the current one. However, it does not remove the current process' PCB from the ready list. In addition, the processor saves the contents of the preempted process' registers before starting to execute the new process.

As the above explanation shows, NOTIFY does not always interrupt the currently executing procedure. However, it does always make a change in the specified semaphore.



NOTIFY Instructions, Part 1
Figure 9-5a



NOTIFY Instructions, Part 2
Figure 9-5b

DISPATCHER

The operations performed by the PXM are mostly governed by the dispatcher. This microcoded routine is responsible for:

- Deciding which process to run next
- Assigning that process a register set
- Managing the register file, including saves and restores
- Turning the process timer on and off

The section Dispatcher Operation below describes the details of the dispatcher's actions.

REGISTER FILES

The number of register files varies according to the processor as shown below.

- For 6350 and 9750 to 9955 II: 8 register files
- For 2350 to 2755, 9650, and 9655: 11 register files

Each register file contains 32 32-bit registers that each have a high half and a low half. Tables 9-2 and 9-3 show the allocation of the register files and the absolute memory locations each occupies. (Appendix B discusses the register files of the earlier systems listed on page 1-1.)

Table 9-2
Register File Allocation for 6350 and 9750 to 9955 II

Register File	Absolute Locs	Use
RF0	0 to '37	Microcode scratch and system registers (set 1)
RF1	'40 to '77	32 DMA channels
RF2	'100 to '137	User register set 2
RF3	'140 to '177	User register set 3
RF4	'200 to '237	User register set 4
RF5	'240 to '277	User register set 5
RF6	'300 to '337	Microcode scratch and system registers (set 2)
RF7	'340 to '377	Spare register set

Note to Table 9-2

The four user register sets listed in this table are called user register sets 2 to 5 to correspond with their register file numbers RF2 to RF5.

Table 9-3

Register File Allocation
For the 2350 to 2755, 9650, and 9655

Register File	Absolute Locs	Use
RF0	0 to '37	Microcode scratch and system registers (set 1)
RF1	'40 to '77	32 DMA channels
RF2	'100 to '137	User register set 2
RF3	'140 to '177	User register set 3
RF4	'200 to '237	User register set 4
RF5	'240 to '277	User register set 5
RF6	'300 to '337	User register set 6
RF7	'340 to '377	User register set 7
RF8	'400 to '437	User register set 8
RF9	'440 to '477	User register set 9
RF10	---*	Microcode scratch and system registers (set 2)

*For system use only.

Note to Table 9-3

The eight user register sets in this table are called user register sets 2 to 9 to correspond with their register file numbers RF2 to RF9.

User Register Files

Table 9-4 defines the register mnemonics used in the user register sets. Table 9-5 shows the format of a user register set for V, I, R, and S modes. All user register sets have the same structure.

Table 9-4
User Register Set Mnemonics*

Mnem	Name	Mnem	Name
A	Accumulator	FLR1	Field length register 1
B	Double-precision long accumulator extension.	GR0	General register 0
DTAR0	Descriptor table address register 0.	GR1	General register 1
DTAR1	Descriptor table address register 1	GR2	General register 2
DTAR2	Descriptor table address register 2	GR3	General register 3
DTAR3	Descriptor table address register 3.	GR4	General register 4
E	Accumulator extension for MPL and DVL	GR5	General register 5
FAC	Floating-point accumulator (R and V modes)	GR6	General register 6
FAC0	Floating-point accumulator 0 (I mode)	GR7	General register 7
FAC1	Floating-point accumulator 1 (I mode)	KEYS	Keys
FADDR	Fault address register	L	Double-precision accumulator
FARO	Field address register 0	LB	Link base register
FAR1	Field address register 1	MODALS	Modals
FCODE	Fault code register	OWNER	PCB address of the process that owns the register contents
FLRO	Field length register 0	PB	Procedure base register
		S	Stack, alternate index
		SB	Stack base register
		X	X index register
		XB	Auxiliary base register
		Y	Y index register

* An H appended to a register mnemonic refers to bits 1 to 16 of that register; an L so appended refers to bits 17 to 32.

Table 9-5
User Register File Structure*

Reg Num**	V Mode	I Mode	S, R Modes	Comments
0	---	GR0	---	
1	---	GR1	---	
2	L,A,B	GR2	A, B (1,2)	A occupies L bits 1 to 16; B occupies L bits 17 to 32.
3	E	GR3	---	
4	---	GR4	---	
5	Y	GR5	S (3)	S and Y are 16 bits long.
6	---	GR6	---	
7	X	GR7	X (0)	X is 16 bits long.
'10 to '11	FARO, FLRO,	FARO, FLRO, FACO	('13)	Discussed in the <u>Floating- Point</u> section in Chapter 6. Also, important cautions appear in the section <u>Overlap Between Floating- point and Field Registers</u> of Chapter 9.
'12 to '13	FAR1, FLR1, FAC	FAR1 FLR1, FAC1	FAC (4,5,6)	
'14	PB	PB	PB	These are base registers
'15	SB	SB	SB ('14,'15)	discussed in Chapter 3.
'16	LB	LB	LB ('16,'17)	Their format appears in
'17	XB	XB	XB	Figure 3-1.
'20	DTAR3	DTAR3	DTAR3 ('10)	The format appears in Figure
'21	DTAR2	DTAR2	DTAR2	4-9. The segment group of
'22	DTAR1	DTAR1	DTAR1	each DTAR is as follows.
'23	DTARO	DTARO	DTARO	DTARO: Segs 0 to 1023 DTAR1: Segs 1024 to 2047 DTAR2: Segs 2048 to 3071 DTAR3: Segs 3072 to 4095
'24	KEYS, MODALS	KEYS, MODALS	KEYS, MODALS	The keys occupy bits 1 to 16; the modals occupy bits 17 to 32. The keys format appears in Figures 5-3 and 5-4. The modals format appears in Figure 5-2.
'25	OWNER	OWNER	OWNER	Discussed in Chapter 9.
'26	FOODE	FOODE	FOODE ('11)	Discussed in the Fault
'27	FADDR,	FADDR	FADDR ('12)	section of Chapter 10.
'30	TIMER	TIMER	TIMER	Discussed in the <u>Process Interval Timer</u> section later in Chapter 9.
'31 to '37	---	---	---	

* All registers are 32 bits long unless otherwise indicated. Numbers in parentheses represent address traps listed in Table 3-13.

** Register number within a user register file.

The '25th location in each user register set specifies OWNER, the address of the PCB associated with the process that owns the register set. Bits 1 to 16 of OWNER specify OWNERH, the number of the segment containing the ready list and the PCBs. Make sure that OWNERH contains the proper value in all user register sets before entering process exchange mode.

Overlap Between Floating-point and Field Registers

Floating-point accumulators and field registers occupy the same locations in user register files, as noted in Chapter 6. The precise degree of overlap varies among the 50 Series processors since the 16-bit exponent of the floating-point accumulator can be in one of several places. The guaranteed overlap between the floating-point and field registers is limited to only the following: field register 0 and floating-point accumulator 0 occupy RF2 '110, '111; field register 1 and floating-point accumulator 1 occupy RF2 '112, '113. (These registers occupy similar locations in RF3 through RF9.)

In addition, some processor models (such as the earlier processors 750 and 850) store the floating-point accumulator and the field registers in separate places. Due to this situation, special tracking hardware enforces the following guarantees for all 50 Series processors.

- Floating-point accumulators 0 and 1 are handled independently. Changes to one do not affect the state of the other.
- Any floating-point update sets the register acted upon to the floating-point type.
- Any field update sets the register acted upon to the field type.
- Type-specific stores save from only their type of accumulators. For example, FST looks always to the floating-point storage.
- Type-independent operations take information from the most recently used register type. Examples: LDLR, RSAV, process exchange register save.
- Type-independent operations place information into both types of registers. Examples: STLR, RRST, PX register restore.

The consequences of these guarantees for all 50 Series processors are:

- A DFST following an EAFA may or may not store the EAFA information, depending on an intervening interrupt. (Information is captured if an interrupt occurs.)
- Using anything to save these registers is guaranteed under only the following two circumstances.

- Only one type is used. Examples: DFLD, DFST; EAFA, STFA.

or

- Type-independent instructions are used for the save. Examples: LDLR or RSAV.

DMA Channel Register File

The DMA register file, RF1, contains 32 channel registers. Table 9-6 shows the format of this register file.

Table 9-6
DMA Register File (RF1) Format

Loc	Contents	Loc	Contents
'40	DMA cell 00	'60	DMA cell 20
'41	DMA cell 01	'61	DMA cell 21
'42	DMA cell 02	'62	DMA cell 22
'43	DMA cell 03	'63	DMA cell 23
'44	DMA cell 04	'64	DMA cell 24
'45	DMA cell 05	'65	DMA cell 25
'46	DMA cell 06	'66	DMA cell 26
'47	DMA cell 07	'67	DMA cell 27
'50	DMA cell 10	'70	DMA cell 30
'51	DMA cell 11	'71	DMA cell 31
'52	DMA cell 12	'72	DMA cell 32
'53	DMA cell 13	'73	DMA cell 33
'54	DMA cell 14	'74	DMA cell 34
'55	DMA cell 15	'75	DMA cell 35
'56	DMA cell 16	'76	DMA cell 36
'57	DMA cell 17	'77	DMA cell 37

Directly Addressing a Register Set

To address the register file directly, you must use the LDLR/STLR instructions. For more information, refer to the descriptions of LDLR and STLR in the Instruction Sets Guide. Some register set locations can be addressed as memory locations in some addressing modes as well. See the Address Traps section in Chapter 3 for more information on this topic.

Microcode Register Files

RF0, RF6, and RF10 are reserved for microcode use. These registers can hold temporary data, control information, or other such items for the microcode to use. Some are defined for microdiagnostic use. Tables 9-7 through 9-10 define the microcode register files. (See Appendix B for a definition of the microcode register file for the earlier processors listed on page 1-1.)

Table 9-7

Microcode Register File Set 1, RF0,
For the 6350 and 9750 to 9955 II

Loc	Contents	Loc	Contents
0	TR0	'20	RMA SAVE
1	TR1	'21	---
2	TR2	'22	PARREG1
3	TR3	'23	PARREG2
4	TR4	'24	DSWPARITY2*
5	TR5	'25	PB SAVE
6	TR6	'26	SYSREG1
7	TR7	'27	DSWPARITY
'10	FRO32, TR8	'30	PSWPB
'11	TR9	'31	PSWKEYS
'12	FR132, TR10	'32	PLA, PPA
'13	TR11	'33	PLB, PPB
'14	REOIV, UCSADDR	'34	DSWRMA
'15	RDSAVE	'35	DSWSTAT
'16	CFF00, COOFF	'36	DSWPB
'17	RATMP	'37	RSAPVTR

* For the 9750 to 9955 II, location '24 contains PARREG3.

Table 9-8

Microcode Register File Set 2, RF6,
For the 6350, and 9750 to 9955 II

Loc	Contents	Loc	Contents
'300	DGR0/STLBRF1	'316	C800008 (6350); DGR16
'301	DGR1/STLBRF2	'317	DGR17
'302	DGR2/RDMX1	'320	MINUS1
'303	DGR3/RDMX2	'321	ONE32
'304	DGR4/RDMX3	'322	IUART, KMASK
'305	DGR5/MINKBUF1	'323	C3FF, C3F
'306	DGR6/MINKBUF2 (6350); RSSAV (9955, 9955 II)	'324	C8000
'307	DGR7	'325	CODOD, CBOBOL
'310	DGR10	'326	C9C00, C0080
'311	DGR11	'327	CB1EO, PICSTAT (6350)
'312	ISDIAGRF (6350); DGR12 (see note)	'330	C6666
'313	MCDIAGRF (6350); FF80 (9955, 9955 II); DGR13 (975x)	'331	C1OK, ACK2
'314	MCTEMP1 (6350); DGR14	'332	DP6 (6350); FERRET6
'315	MCTEMP2 (6350); DGR15	'333	DP5 (6350); FERRET5
		'334	DP4 (6350); FERRET4
		'335	DP3 (6350); FERRET3
		'336	DP2 (6350); FERRET2
		'337	DP1 (6350); FERRET1

Notes to Table 9-8

For the 9955 and 9955 II, location '312 is nonzero when a recoverable machine check has occurred. In this case, the value and meaning of bits 29 to 32 of '312 are the same as those in bits 29 to 32 of the DSWPARITY for the 9955 and 9955 II. (Chapter 10 discusses the recoverable machine check as well as the DSWPARITY.) The meanings of bits 29 to 32 are as follows.

Bits	Meaning
29	If 1, the S unit detected an error.
	Bits 30 to 32 describe the error.
30 to 32	For the 9955 and 9955 II only:
	000: no error
	001: LPID out of STLBI in error
	010: LBPA out of STLBI in error
	011: LBVA out of STLBI in error
	100: ARR out of STLBI in error
	101: cache index
	110: cache data high side
	111: cache data low side

Table 9-9
Microcode Register File Set 1, RFO, for the 2350 to 2755, 9650, 9655

Loc	Contents	Loc	Contents
0	TR0	'20	ONE32
1	TR1	'21	PBSAVE
2	TR2	'22	RDMX3
3	TR3	'23	RDMX4
4	TR4	'24	C377
5	TR5	'25	MINUS1
6	TR6	'26	IREGSET,CHKREG
7	TR7	'27	DSWPARITY
'10	RDMX1	'30	PSWPB
'11	RDMX2	'31	PSWKEYS
'12	USCADDR, REOIV	'32	PPA
'13	RSGT1	'33	PPB
'14	RSGT2	'34	DSWRMA
'15	REOC1	'35	DSWSTAT
'16	REOC2	'36	DSWPB
'17	TEMPCAC	'37	RSVPTR

Table 9-10
Microcode Register File Set 2, RF10, for the 2350 to 2755, 9650, 9655

Loc	Contents	Loc	Contents
'500	DECO0	'521	ADRREG2
'501	DECO1	'522	ADRREG
'502	DECO2	'523	LIGHTS, INTVEC
'503	DECO3	'524	QPTR, BYTFLG
'504	DECO4	'525	WSLFLG
'505	DECO5	'526	RDMX5; SCR26 (2755 only)
'506	DECO6	'527	UMASK1, SCR27L;
'507	DECO7		SCR27 (2755 only)
'510	DEC10	'530	UMASK2, SCR30L;
'511	REOC3		SCR30 (2755 only)
'512	TMRSVE	'531	URDRXH, SCR31L;
'513	CTRLEYTE, QFDIDX;		SCR31 (2755 only)
	SCR13 (2755 only)	'532	BFR04
'514	CMDEYTE, SCR14L;	'533	DSSW
	SCR14 (2755 only)	'534	RSTLB1
'515	EXP32	'535	RSTLB2
'516	SSN; RDMX5 (2755 only)	'536	RSTLB3
'517	SWITCHES, PICSTAT	'537	RSTLB4
'520	WWADTR; IDREG (2755 only)		

PROCESS INTERVAL TIMER

The process interval timer is a 48-bit number that represents the time that has passed since this process began executing (or, for system processes, the time since cold start). The timer represents time in units of 1.024 milliseconds. Bits 1 to 42 of the timer represent the time; bits 43 to 48 are reserved for future use.

Four PCB locations contain timer information; the TIMER register is in the User Register File at relative location '30. Table 9-11 describes the PCB locations and their contents.

Table 9-11
Timer Control Information

PCB Loc	Name	Contents
'10 to '11	Elapsed Timer	Total time used by this process in units of 1.024 msec.
'16	Interval Timer High	Copy of TIMERH from location '30 in the current register set. This value is the two's complement of the number of 1.024 msec intervals left before the end of the time slice.
'17	Interval Timer Low	Bits 1 to 10 contain a copy of TIMERL from location '30 in the current register set. This value is the amount of process time used in units of one usec. Bits 11 to 16 are reserved.

Timers are accurate to the microsecond. (For some earlier processors listed on page 1-1, however, timers are accurate to the millisecond; see Appendix B.) The process timer represents the amount of time that has passed in the current timeslice. The interval timer contained in the register file locations represents the amount of time remaining in this timeslice. Figure 9-6 shows how to use these two values to calculate the time that has passed since the last reset.

	LDL	ET	/* load L with value of ET	
	STL	SET	/* save the current value of ET at location SET	
	LDA	RESET	/* load A with the reset value	
	RTS		/* reset the timeslice	
	IMA	CURRTS	/* save the reset value in CURRTS, load A with	
			/* previous reset value	
	SUB	RESET	/* find difference between new, old reset values	
	TCA		/* form two's complement of contents of A	
	PIDA		/* position for addition	
	ADL	ET	/* add difference of reset values to contents of ET	
	SBL	SET	/* subtract old value of ET from contents of L	
			/* L now specifies the time that has passed since	
			/* the last timer reset.	

Microsecond Timer Example
Figure 9-6

Two instructions, RTS and STTM, manipulate the process timer. Table 9-12 describes these instructions. (See Appendix B for a discussion of RTS, STTM, and the earlier processors listed on page 1-1.)

Table 9-12
Process Timer Instructions

Mnem	Name	Modes	Description
RTS	Reset Timeslice	V,I	Adds the contents of A, the interval timer, and the elapsed timer and stores the result in the elapsed timer. Loads the contents of A into the interval timer.
STTM	Store Process Time	V,I	Stores the contents of the process timer into memory.

DISPATCHER OPERATION

As mentioned earlier, the dispatcher governs most of the actions of the PXM. These can be divided into the following steps:

1. Turning off the process interval timer
2. Choosing a process to run
3. Selecting a user register set for that process
4. Turning the process interval timer back on

The paragraphs below elaborate on each of these steps.

Step 1. Turning off the Process Interval Timer

As soon as the dispatcher begins to execute, it turns off the process interval timer. This timer is located in bits 1 to 26 of location '30 in the current register set. (See Appendix B for this timer's location in the earlier systems listed on page 1-1.) It contains a negative number specifying the amount of time left in the current time slice. On each tick, this negative value is incremented by 1; when the incremented value reaches 0, bit 16 of the PCB's abort flags is set to 1. When a process is dispatched and bit 16 of the abort flags in its PCB is set to 1, a process fault is taken. The abort is only effective after a process is stopped. This stoppage must be guaranteed by some high priority, high frequency process usurping the machine. In PRIMOS, the clock process (and frontstop on multi-stream processors) perform this function.

Step 2. Choosing the Next Process to Run

PCBA, contained in PPA, holds information about which process the dispatcher should dispatch next. When the dispatcher is first activated, it checks PCBA; if PCBA contains a nonzero value, it specifies a valid PCB and the dispatcher will dispatch the associated process.

If PCBA contains zero, it is invalid and the dispatcher checks PPB for a nonzero value. If PPB is valid, the dispatcher will dispatch that associated process.

If PPB is invalid, the dispatcher must scan the ready list for the PCB of the next process to dispatch. The scan begins at the level specified by Level A in PPA. If the dispatcher finds a PCB, it changes Level A to reflect the level of the found PCB and dispatches that process next. If it finds no PCB, the ready list is empty and the dispatcher idles.

Step 3. Manipulating User Register Sets

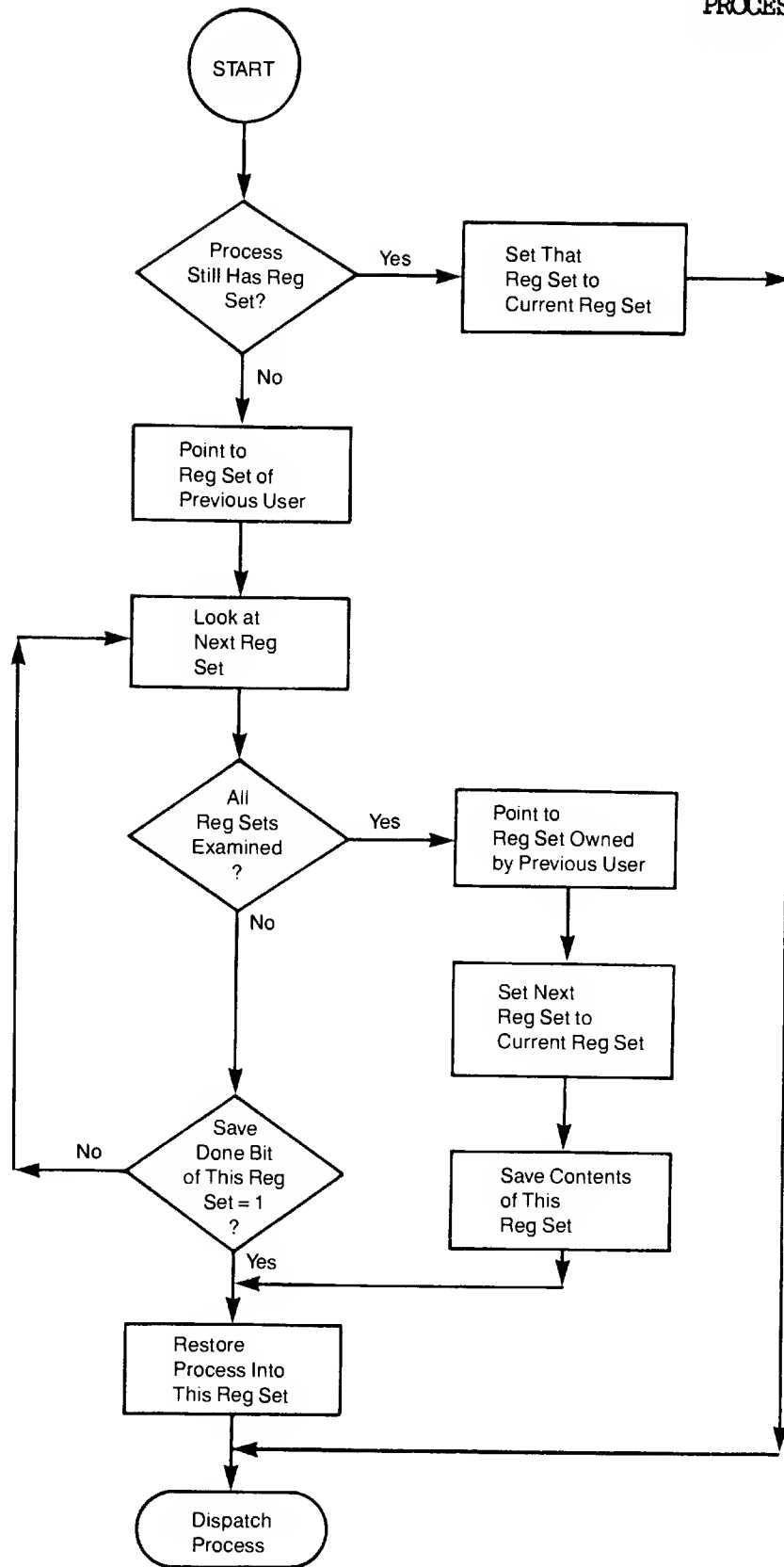
Once the dispatcher has identified the next process to dispatch, it must allocate a user register set to the process. Since there are only a finite number of register sets, the dispatcher may have to swap one register set for another; the new process will require a register set other than that used by the last process. Figure 9-7 shows the allocation algorithm that the dispatcher uses for all systems but the earlier processors listed on page 1-1. The text in this section elaborates on the figure. (Appendix B presents the allocation algorithm for these earlier systems.)

The dispatcher first checks whether the process to be dispatched owns the current register set. It looks at the contents of bits 17 to 32 of OWNER (location '25 in the current register file). These specify the address of the PCB whose associated process owns that register set. If OWNERL specifies the address of the PCB associated with the next process to run, then this process owns the current register set. The dispatcher makes no changes in the current register set before dispatching the next process.

If OWNERL specifies the address of some other PCB, the next process to be dispatched does not own the current register set.

The dispatcher reads the contents of offset 5 in the PCB associated with the next process to run to find the number of the register set this process used last. The dispatcher checks OWNER in the register set specified by PCB offset 5 to see if the next process to run owns this register set. If it does, the dispatcher must make this register set the current one. If the next process to run does not own the last register set it used, the dispatcher must choose one for it. It increments the number of the current register set by 1 (modulo 4 for the 6350 and 9750 to 9955 II, and modulo 8 for the 2350 to 2755, 9650, and 9655) to form the number of the new register set, then makes this register set the current one.

3A. The Save Done Bit: In the case where the process does not own the current register set, the dispatcher must load the values of the new process' registers into the current register set. Before it can do this, it must determine whether it must save the old contents of the current register set. Bit 16 of the keys contains the Save Done bit. If this bit contains a 0, the dispatcher must save the old contents of the current register set before restoring the new process to run. After the save, the dispatcher loads the new data into the current register set, resets bits 15 and 16 of the keys (the In Dispatcher bit and the Save Done bit) to 0, and loads the program counter with the contents of PB.



Register Set Allocation Algorithm
Figure 9-7

If the Save Done bit contains a 1, the old contents of the current register set have been saved in the PCB and register file memory locations, so no further save needs to be done before the new data is loaded. After loading the registers, the dispatcher resets bits 15 and 16 of the keys and loads the program counter from PB.

3B. Saving the Current Register Set: When the dispatcher must save the current register set before loading in new data, it saves only the registers that contain nonzero values. The contents of these nonzero registers are packed together and loaded into the save area. The save mask determines which registers have had their contents saved and the exact location of those contents in the PCB.

Only the currently active register set contains valid information in the modals field. Whenever the processor switches register sets, the microcode automatically copies the contents of the current modals field into the new register set.

Step 4. Turning On the Process Interval Timer

The last thing the dispatcher must do before dispatching a process is to turn on the process interval timer. The dispatched process begins execution immediately after.

FETCH CYCLE TRAPS

At various points during dispatcher execution, the processor checks for fetch cycle traps, to allow the system to handle external interrupts. For more information about this topic, refer to Chapter 10, INTERRUPTS, FAULTS, CHECKS, AND TRAPS.

SUMMARY

This chapter described the actions that occur during process exchange for all single-stream processors.

10

Interrupts, Faults, Checks, and Traps

Most of the time, the processor executes instructions contained in one process, then goes on to those contained in another. At some point, however, another part of the system may require service; when this happens, the processor has to break the flow of control within the currently running process and service whatever has interrupted. This chapter describes the types of breaks that can occur, and how the 50 Series processors service them.

BREAKS

Breaks in execution can be caused by four events:

- An interrupt
- A fault
- A check
- A trap

The first three types of events are breaks in software execution. The last, the trap, is a break in microcode execution.

The way in which the processor services a break depends on its type and on the current process exchange mode of the machine. When the PKM is disabled, the processor handles all software breaks in the same way. Interrupts, checks, and faults all vector through a dedicated Sector 0 location to reach their handlers.

When the PKM is enabled, the processor handles each software break with a different protocol. Table 10-1 defines the software breaks and briefly describes the protocols that the machine uses to service them.

Microcode breaks are handled differently. When a trap occurs, it may cause a software break, which the processor services to clear the microcode break. If no software break is necessary, the processor handles the microcode break in a fashion transparent to the currently executing process.

Table 10-1
Summary of Software Breaks

Break	Definition	How Serviced
Interrupt	The processor receives a signal from an external device requiring service.	The currently executing software does not usually cause an interrupt. Code especially designed for the purpose services the interrupt outside the context of the currently executing process.
Fault	The currently executing software requires software intervention.	The currently executing software usually handles a fault by mirroring a procedure call to fault code. This code services the fault within the context of the current process.
Check	The processor detects an internal consistency problem requiring software intervention, such as an integrity violation, a reference to a nonexistent memory module, or a power failure.	As with interrupts, code designed especially for the purpose services the check outside the context of the currently executing process.

INTERRUPTS

Interrupts take the form of external interrupts. As mentioned above, actions depend on whether the PXM is enabled or disabled. (The earlier processors listed on page 1-1 also have memory increment interrupts as discussed in Appendix B.)

External Interrupts, PXM Disabled

If an external interrupt occurs when the PXM is disabled, the processor uses the address specified by the controller (vectored interrupt mode) to build a vector as described below. This vector points to the interrupt response code (IRC).

The I/O bus address lines received by the CPU are interpreted and used differently based on whether mapped I/O is enabled or not. (The controller has no knowledge of which of these modes the CPU is operating in or of the CPU type.)

The CPU reaches the IRC by performing an indirect JST through the address that is calculated in the ways shown below.

When the CPU is not in mapped I/O mode, the processor forms bits 1 to 16 of the address from the contents of the location specified by 0 concatenated with BPA 99, 00, and 1 to 16. In this mode, the CPU reads physical memory.

When the CPU is in mapped I/O mode, the processor forms bits 17 to 32 of the virtual address from the contents of the location specified by 0 concatenated with BPA 1 to 16. In this mode, the CPU reads Segment 0, offset BPA 1 to 16.

Whether the CPU is in mapped I/O or not, the following rules apply. If the address is 0, then a halt occurs and the program counter contains the address that it had when the interrupt was processed. Otherwise, the following actions occur in this order: bits 1 to 16 of the address are cleared; the contents of the program counter are saved at the location specified by the 32-bit address; this address is incremented by 1 and loaded in the program counter; interrupts are inhibited. Note that the Segmentation Modal determines whether the program is in physical memory or in Segment 0.

Interrupts are disabled when the IRC begins execution, but all other keys and modals remain unchanged. In vectored mode, the IRC must clear the active interrupt before reenabling interrupts. After the clear, the IRC reenables interrupts, saves the current contents of any register it intends to use, and completes the rest of its operation. When it is done, it transfers control back to the location whose address is contained in the first IRC location.

External Interrupts, PXM Enabled

If an external interrupt occurs when the PXM is enabled, the processor uses the address sent by the controller as a 16-bit offset into a segment as described below.

When PXM is enabled, the processor assumes that segmentation is also enabled. The CPU forms the 32-bit address from 4 concatenated with BPA 1 to 16. (This is Segment 4, offset BPA 1 to 16.) The contents of the program counter are saved in PSWPB, and the contents of the keys are saved in PSWKEYS. (PSWPB and PSWKEYS are the phantom code scratch registers.) The microsecond timer is turned off, and the keys are set to reflect 64V mode. Interrupts are inhibited, the program counter is loaded with the 32-bit address. The IRC (called the immediate IRC, or phantom interrupt code) begins to execute.

Phantom Interrupt Code: Phantom interrupt code gives the processor a chance either to perform a trivial task to service the interrupt, or, as happens most often, merely to notify the real interrupt handler. It is usually only a few instructions long. An example of what the phantom interrupt code might look like is shown in Figure 10-1.

Code Purpose	Code Sequence	Comments
Perform trivial task	STA address	Save A register.
	EIO address	Read a 16-bit quantity from a device.
	ABQ address	Add entry to the bottom of a queue.
	LDA address	Restore A register.
	IRTC	Clear interrupt from I/O bus, enable interrupts, and return to normal execution.
Notify	INBC address	Notify a process, clean up the I/O bus, and enable interrupts.

Sample Phantom Interrupt Code Sequences
Figure 10-1

Some restrictions govern phantom interrupt code. Since it has no PCB that PPA can reference, it does not belong to a process. Also, phantom interrupt code saves only PB and the keys. If another interrupt were to occur before the phantom interrupt code completed service to a previous interrupt, the contents of PSWPB and PSWKEYS would be

overwritten, destroying information about the first interrupt. Therefore, interrupts must remain inhibited until the phantom interrupt code completes. Furthermore, the microsecond timer is suspended during phantom interrupt code, so the time spent in this code is not accounted for.

Because of these restrictions, the phantom interrupt code can completely service only very simple interrupts. If more complete service is required, the phantom interrupt code only turns off the controller's interrupt mask, clears the currently active interrupt, and notifies the process that will perform the action requested by the interrupt.

Returning From an External Interrupt: When the IRC completes, it issues either an interrupt return (IRTN, IRTC) if completely finished or an interrupt notify (INEN, INEC, INBN, INBC) to notify the real interrupt handler. The IRTN restores the keys and PB with the saved contents of PSWKEYS and PSWPB, respectively, and enables interrupts, leaving the machine state as it was before the interrupt. (Restoring the keys also restores the addressing mode to what it was before the interrupt.) The interrupt notify (INOTIFY) instructions put the machine back to the pre-interrupt state by reloading PB and the keys from PSWPB and PSWKEYS, enabling interrupts, and executing the appropriate notify instruction. This allows the process exchange mechanism to work as if the phantom interrupt code did not happen, returning to the code originally interrupted.

All phantom interrupt code sequences must clean up the I/O bus by issuing a CAI signal before interrupts are reenabled. This can be done by using IRTC, INEC, or INBC instructions as appropriate. Alternatively, a CAI can be issued in the IRC before exiting phantom interrupt code through an IRTN, INEN, or INBN instruction as appropriate.

FAULTS

Faults occur when software tries to perform an action that cannot complete without special help. Examples of faults are page faults (where a reference is made to a page not currently loaded in physical memory) and stack overflow or underflow. In all, there are twelve classes of faults that can occur. Table 10-2 introduces these classes and their subdivisions.

Table 10-2
Fault Classes

Fault	50 Series Systems
RXM	Restrict mode violation
Process	Abort flags content does not equal 0 in PCB on dispatch
Page	Page fault (page not in memory)
SVC	Supervisor call (superceded by direct entry calls)
UII	Unimplemented instruction
ILL	Illegal instruction
Semaphore*	Semaphore overflow or underflow
Access	Violation of segment access rights
Arithmetic	All FLEX, DEX, and IEX (arithmetic exceptions)
Stack	Stack overflow/underflow
Segment	1: Segment number too big (SDT too short) 2: Missing segment (SDW fault bit set)
Pointer	Fault bit in pointer set

* Unavailable for the earlier systems listed on page 1-1.

Fault Handler

The software routine that services faults is called the fault handler. It is made up of two parts: a group of entrances (one entrance for each type of fault) and a common fault routine. When a fault occurs, execution begins at the entrance for that fault type. The entrance microcode sets up conditions applicable to the fault, then transfers control to the common handler. This arrangement provides service for several types of faults while avoiding the expense of many different handlers.

There are four elements in the fault mechanism:

- Four fault vectors
- Four fault tables
- The Call Fault Handler (CALF) instruction
- The concealed stack

The microcode routine uses these four elements to convert faults into procedure calls to the various service routines.

Fault Vectors

The fault vectors occupy locations '62 to '65 and '70 to '73 in the PCB. Each vector contains the address of a fault table. (See Fault Tables, below.) The format of the vectors is identical to that of a 32-bit indirect pointer, as shown in Figure 3-3 in Chapter 3.

The vectors provide a choice of how to handle a particular fault. For example, one process may need to have Ring 0 service a pointer fault, while another process defines its own routines in the current ring to do the service. Since the vectors are located in the process' PCB, different vectors can be specified for processes that need different service. Table 10-3 describes the PCB locations that contain the vectors.

Table 10-3
PCB Fault Vector Locations

PCB Loc	Contents
'62 to '63	Ring 0 fault vector
'64 to '65	Ring 1 fault vector
'70 to '71	Ring 3 fault vector
'72 to '73	Page fault fault vector

A separate vector is devoted to page faults, even though page faults require Ring 0 service. This allows a system to specify a universal page fault handler to handle all page faults that occur within the system. If a system uses a universal page fault routine, make sure that all page fault vectors for processes currently within the system contain the address of this universal routine, rather than some other Ring 0 routine.

When a fault occurs, the program counter is loaded with the fault vector in the PCB, including the ring number. This means that fault code is not automatically executed in either Ring 0 or the current ring: the code in the fault tables may either weaken the ring or go through a gate to strengthen the ring.

Fault Tables

Each fault vector points to a fault table. Each table contains 12 8-byte entries, each entry corresponding to one of the types of faults. Table 10-4 lists information about the fault table. In this table, the Offset column applies when process exchange is enabled; the Vector location column applies when process exchange is disabled. FCODEH (bits 1 to 16 of FCODE) and FADDR are in each user register file. The Ring column shows the ring number at the time of the fault. The Saved PB column tells whether or not the PB had to be backed up to the instruction having the fault.

The fault table for page faults must always be located in physical memory. A page fault must never result in an unresolved chain of page faults. For these reasons, the fault table for Ring 0 must exist in a defined segment. If it does not, it is possible to have an infinite number of segment faults occurring recursively, since the Ring 0 fault table for each fault never references a valid segment.

Table 10-4
Fault Information

Fault	Num	Offset	Vector Loc	FCODEH	FADDR	Ring	Saved PB
RXM	0	0	'62	0	Addr1	Cur	Backed
Process	1	4	'63	ABFLAGS	---	0	Cur
Page	2	'10	'64	0	Addr2	0	Backed
SVC	3	'14	'65	0	---	Cur	Cur
UII	4	'20	'66	Cur RPL	Addr1	Cur	Backed
Semaphore*	5	'24	'67	Underflow	Addr of	0	Backed
				\$0;	Sema-		
				Overflow	phore		
				\$1			
ILL	'10	'40	'72	Cur RPL	Addr1	Cur	Backed
Access	'11	'44	'73	0	Addr3	0	Backed
Arith.	'12	'50	'74	See Table	Addr4	Cur	Cur
				10-10			
Stack	'13	'54	'75	0	Addr5	Cur	Backed
Segment	'14	'60	'76	DTAR: 1;	Addr3	0	Backed
				SDW: 2			
Pointer	'15	'64	'77	PCL:	Pt Adr	Cur	Backed
				'100000;			
				Else the			
				high half			
				of the			
				faulting			
				pointer			

Notes to Table 10-4

* Unavailable for the earlier processors listed on page 1-1.

Addr 1: Find the effective address specified by the instruction.

Addr 2: Virtual address pointing to the missing page. The low order 9 bits of the address need not be the offset used by the reference causing the fault.

Addr 3: FADDR contains the address generating the fault.

Addr 4: See Table 10-10.

Addr 5: Pointer to the last valid stack extension segment. (Can be the root segment, offset 2.)

The CALF Instruction

Each entry in the fault table can contain any type of instruction, but usually the instruction is either a HLT or a CALF instruction. When the entry contains a HLT, the machine stops every time the fault corresponding to that entry occurs.

When the fault table entry contains a CALF instruction, the format of the entry is as shown in Table 10-5. Bytes 3 to 6 of CALF contain a pointer to the ECB of a fault routine. CALF uses this pointer to transfer control to the fault routine as if the transfer were a normal procedure call. The advantages of this are described in Servicing a Fault, below.

Table 10-5
Format of Fault Table Entries

Byte	Contents
1 to 6	CALF instruction. Bytes 3 to 6 contain a pointer to the ECB of a software fault handler.
7 to 8	Reserved.

CALF performs a normal procedure call where no arguments are expected by the callee. If the callee's ECB specifies arguments, then dummy arguments are substituted and loaded into the stack frame. See Chapter 8 for information about dummy arguments.

The rest of this section describes how a fault is handled if the associated fault vector contains a CALF instruction.

The Concealed Stack

When a fault occurs, the state of the system at the time of the fault must be saved before the fault can be serviced. The processor uses the concealed stack to save information about the system state at the time of a fault.

Information is stored in the concealed stack in frames. Each frame contains 12 bytes of information, as shown in Table 10-6.

Table 10-6
Concealed Stack Frame Format

Offset	Contents
0 to 1	Program counter (segment #/offset)
2	Keys
3	Fault code high (bits 1 to 16)
4 to 5	Fault address (segment #/offset)

Six bytes of the PCB keep track of the concealed stack frames. These bytes contain the addresses of the first, last and next available frames in the concealed stack. Table 10-7 describes these locations.

Table 10-7
Contents of PCB Concealed Stack Locations

Loc	Name	Description
'74	FIRST	Pointer to the first frame in the concealed stack.
'75	NEXT	Pointer to the next frame to be used.
'76	LAST	Pointer to the last frame in the concealed stack.
See desc	---	Six 12-byte concealed stack frames that can go anywhere in segment OWNERH + 1 for the 6350; they can go anywhere in segment OWNERH for all other processors.

The processor uses a separate stack for faults to simplify handling chains of faults. Frequently the CALF instruction for one fault can generate another fault, such as a segment fault, when it tries to call the fault handler. The CALF for this fault may in turn cause another fault, and so on. Instead of using the current segment's stack to contain the information about all of these faults, the concealed stack is used. Since pointers to the concealed stack are located in the PCB, the fault handler can easily access it, and there is no danger of using data from anything other than a fault frame. (The architecture does not support page faults when searching for a PCB or a concealed stack.)

If a chain of faults occurs, the processor services them in reverse order: the last fault to occur is the first to be serviced.

The concealed stack can accommodate a chain of up to n faults, ($n = 6$ in PRIMOS), one fault per concealed stack frame. Make sure that the concealed stack contains enough frames to allow for the longest chain of faults that can occur. Since the concealed stack is circular, if one more fault occurs than there are concealed stack frames, the frame for the latest fault will overwrite that of the first fault. For example, suppose the concealed stack contains only four frames, and the chain of faults that occurs is:

pointer (link) fault->segment fault->stack fault->segment->page fault

The frame for the page fault overwrites that of the link fault frame. The concealed stack no longer contains the proper information about the link fault frame, so the link fault will never be serviced.

Servicing a Fault

As with interrupts, the type of fault service that the processor performs depends on whether the PXM is enabled or not. If the PXM is disabled, it handles all faults in the same way. It saves the contents of the program counter into DSWPB, disables interrupts for one instruction (if the fault is ultimately to be serviced by a Ring 0 handler only), and jumps indirectly (JST) through a fault vector to the appropriate handler.

If the PXM is enabled, the processor must perform a more complex routine:

1. Set up a concealed stack frame.
2. Change the addressing mode to 64V.
3. Select a fault vector.
4. Set PB so that it points to the proper fault table entry.

When a fault occurs, the processor identifies the fault's type by indexing into the fault table. (See Table 10-4.) After identifying the type of fault, the processor uses NEXT to load the next available concealed stack frame with information about the fault. It updates NEXT to point to the next available frame, then sets the machine addressing mode to 64V (if necessary), and references the appropriate fault vector.

The fault vector contains the starting address of a fault table. The processor adds the offset corresponding to the type of fault to this starting address to form the address of a table entry. This entry

contains a CALF instruction that points to the ECB of a fault routine. If the fault is ultimately to be serviced by a Ring 0 fault handler, interrupts are disabled for one instruction to allow the CALF instruction to execute. If a handler in another ring is to service the fault, no such interrupt disable occurs.

When the CALF instruction begins to execute, it allocates a stack frame on the current segment's stack and loads it with the information shown in Table 10-8. CALF gets much of this information from the current concealed stack frame.

After loading the concealed stack frame into the current segment's new procedure stack, CALF pops the most recent frame from the concealed stack and sets the flag word to 1. Control is transferred to the entrance specified in the ECB.

Table 10-8
Format of CALF Stack Frame

Offset	Contents
0	Flag bits. CALF sets this to 1.
1	Stack root segment number.
2 to 3	Return pointer. This is the value of PB found in the current concealed stack frame.
4 to 5	SB. This value is unchanged.
6 to 7	LB. This value is unchanged.
8	Keys. This is the value of the keys found in the current concealed stack frame.
9	Address of the location following the call.
10	Fault code.
11 to 12	Fault address.
13 to 15	Reserved.

When the handler completes, the PRTN instruction transfers control to the location specified in offsets 2 to 3 in the current segment stack frame. Offsets 2 to 3 contain the saved PB value shown in Table 10-4. This value and the type of fault that occurred determine the actions of the processor after it completes fault service. (For example, the processor might retry the instruction that caused the fault.)

The ECB specified by the stack frame in the current segment's stack must not specify any arguments. It can be a gate or not.

Summary of Fault Classes

Table 10-4 listed the twelve types of faults. Table 10-9 briefly describes what causes each type.

Table 10-9
Summary of Fault Classes

Fault	Cause	Source of Fault
RXM	Non-ring 0 process tries to execute a restricted instruction when restricted mode is enabled.	Hardware; from microcode independent action code.
Process	Offset 4 in the PCB does not contain 0 upon dispatch.	Dispatch microcode test.
Page	Reference made to page with missing bit reset to 0. This usually indicates that the page is not in physical memory.	STLB update microcode test.
UII	Processor tries to execute an instruction that is not implemented on this machine.	Decode net or microcode branch.
Semaphore*	A semaphore has either overflowed due to many notifies, or has underflowed due to too many waits.	NOTIFY or WAIT microcode.
ILL	Processor tries to execute an illegal instruction.	Decode net or microcode branch.
Access	Reference made to a segment without the proper access rights.	STLB update microcode test.
Arithmetic	Integer, decimal, or floating-point exceptions.	If IEX, hardware; if not, explicit microcode test.
Stack	Stack overflow or underflow has occurred.	PCL microcode.
Segment	Either the specified segment number is too big, or the segment is missing.	STLB update microcode test.
Pointer	The fault bit in the specified pointer is 1, indicating an invalid pointer.	IP processing in ARGT fetch microcode.

* Unavailable for the earlier processors listed on page 1-1.

Arithmetic Exceptions

The arithmetic exceptions (integer, floating-point, and decimal) require more explanation than is given in Table 10-9. These three exceptions determine what type of action occurs when an arithmetic overflow, divide by zero, or other such condition exists. Three bits in the keys select what action should occur:

- Bit 7 in the keys specifies the action that is to occur if a floating-point exception occurs.
- Bit 8 determines the action that should follow an integer exception.
- Bit 11 determines the action that should follow a decimal exception.

When any of these exceptions occur, the processor checks the value of the corresponding bit in the keys. In the case of integer and decimal exceptions, a 0 in the corresponding bit tells the processor only to set CBIT to 1. When the corresponding bit in the keys contains a 1, the processor not only sets CBIT to 1, but also loads three registers -- FCODEH, FCODEL, and FADDR -- with appropriate values, and services the fault.

The processor takes the same actions when a floating-point exception occurs, except that when bit 7 in the keys contains a 1, the processor only sets CBIT to 1. When bit 7 contains a 0, the processor both sets CBIT to 1 and services the exception.

FADDR, FCODEH, and FCODEL are located in the user register file. When the processor loads these registers, FCODEL always contains a '50, which indicates that an arithmetic fault has occurred. FCODEH contains a code that identifies the specific exception that has occurred. FADDR contains a pointer to the instruction that caused the exception, a pointer to the address used by the faulting instruction, or 0. Table 10-10 lists the codes and the faults they indicate.

For the earlier processors listed on page 1-1, see Appendix B for the actions taken when an integer overflow exception occurs.

Table 10-10
Arithmetic Exception Codes

Data Type	Exception Type	FOODEH	FADDR
Single precision floating-point	Exponent overflow	\$100	Address of faulting instruction
	Divide by 0	\$101	Address of faulting instruction
	Store exception on FST instruction	\$102	Memory address used by FST
	INT exception	\$103	Address of faulting instruction
	Intrinsic function exception	\$500	Address of faulting instruction
Double precision floating-point	Overflow or underflow	\$200	Address of faulting instruction
	Divide by 0	\$201	Address of faulting instruction
	Intrinsic function exception	\$600	Address of faulting instruction
Integer	Integer overflow	\$300	0
	Divide by 0	\$301	Address of faulting instruction
Decimal	Decimal overflow*	\$700	Address of faulting instruction
	Divide by 0	\$701	Address of faulting instruction
	Conversion exception	\$702	Address of faulting instruction
Quad precision floating-point	Overflow or underflow	\$800	Address of faulting instruction
	Divide by 0	\$801	Address of faulting instruction
	QINQ exception	\$803	Address of faulting instruction

* See Appendix B for decimal overflow FOODEH contents for the earlier processors listed on page 1-1.

CHECKS

The last section described how problems in a process or procedure cause faults. When problems arise with the state of the system itself, a check occurs. These problems may not be visible to the currently executing procedure, or they may be serious enough to terminate the entire system's operation. There are six types of checks:

- Power failure
- Memory parity error
- Machine check
- Recoverable machine check (6350, 9955, and 9955 II only)
- Missing memory module
- Environmental (2350 to 9955 II only)

The power supply for the system initiates a power failure check when AC power fails. The check indicates that 20 milliseconds of DC power remains before all power is gone.

The memory error checking logic issues a memory parity error check when it detects a memory parity error or an uncorrected error correction code (ECCU) error.

The CPU issues a machine check when it detects an internal parity error or (for the 6350 only) a problem in the microcode. For the 6350, 9955, and 9955 II only, the CPU issues a recoverable machine check when it detects a recoverable parity error in the STLB or cache.

The MCU initiates a missing memory module check when a program tries to access nonexistent physical memory.

Environmental sensors are supported only by the 2350 to 9955 II and consist of a diagnostic/maintenance processor system that supports inputs from the UPS (uninterruptable power supply) system and environmental sensors. This system allows the processor to be brought to an orderly shutdown in the event of such things as an overtemperature or a main AC power loss with messages appearing on the supervisor terminal.

Several types of environmental sensors are listed in Table 10-11. An environmental sensor check uses the same check vector and DSWSTAT as power failures, which are presented later in this chapter. In addition, each environmental sensor that produced the check has its own check code as shown in Table 10-11. An environmental check code is stored in register 26L (CHKREG), which is valid only after a processor check has been issued. (All other checks store 0 in that register.)

Table 10-11
Environmental Check Codes*

Environmental Check	Processors	Code
Power supply temperature	6350	\$00
Processor board temperature	2350 to 9955 II	\$01
Maintenance processor temperature	2350, 2450, and 6350	\$01
Ambient temperature	6350	\$02
Cabinet temperature**	2350 to 2755, and 9650 to 9955 II	\$02
Air flow	2350 to 9955 II	\$04
UPS/Battery backup***	2350 to 9955 II	\$08
Soft system shutdown request	2350, 2450, and 6350	\$10

Notes to Table 10-11

* Environmental sensors are not supported by the earlier processors listed on page 1-1.

** For the 6350, the function provided by the cabinet temperature sensor is combined with the airflow sensor, and the check code is \$04.

*** Battery backup capability is supported only for the 6350.

Power Supply Overtemperature (6350): When an overtemperature condition is detected on the power supply, the maintenance processor initiates an immediate system powerdown that includes powering down the processor.

Processor Board Overtemperature: When an overtemperature condition is detected on the processor board, the maintenance or diagnostic processor initiates an immediate system powerdown that includes powering down the processor.

Maintenance Processor Overtemperature (2350, 2450, 6350): When an overtemperature condition is detected on the maintenance processor, the maintenance processor initiates an immediate system powerdown that includes powering down the processor.

Ambient Overtemperature (6350): When the temperature of the air surrounding the 6350 is too high, the maintenance processor initiates an orderly system shutdown by sending the processor an environmental check code that initiates the PRIMOS system shutdown. The maintenance processor waits for a CPU halted message or for a specified timeout: 5 minutes for the 2350, 2450, and 6350; 10 minutes for the 2550 to 2755, and the 9650 to 9955 II.

Cabinet Overtemperature and Airflow Sensors: When the cabinet temperature is too high or the airflow sensor detects a failure in the cabinet blowers, the maintenance or diagnostic processor initiates an orderly system shutdown by sending the processor an appropriate environmental check code that initiates the PRIMOS system shutdown.

The diagnostic processor waits for a CPU halted message or for a specified timeout (10 minutes for cabinet overtemperature, 1 minute for air blower failure). If an air blower failure occurs while there is more than 1 minute to timeout, the timeout is set to 1 minute.

Uninterruptable Power Supply (UPS) Support: The UPS uses two signals, UPS active and UPS battery low. UPS active means that main AC power has been interrupted. The low battery condition means that several minutes remain before system power is lost.

When the UPS is powering the entire system, including peripherals, and a battery low condition occurs, the diagnostic processor sends a processor check to the CPU, and waits for a CPU halt or for up to 5 minutes before powering down the system.

When a UPS active condition occurs and the UPS is powering only the CPU, memory, and diagnostic processor, the diagnostic processor sends a power failure signal to the processor, causing the processor to log the power failure condition and then halt.

Battery Backup Capability (6350): When the 6350 experiences a power failure, its battery backup capability is able to supply power to the memories, maintenance processor, and memory refresh logic of the CPU.

Soft System Shutdown Request (2350, 2450, 6350): When a user powers down the processor without first shutting down PRIMOS, the maintenance processor shuts down PRIMOS in an orderly manner.

Check Handler

Like the fault handler, the check handler is made up of a group of entrances, one for each type of check, and a common check routine. To service checks, it uses a check header, check vectors, a diagnostic status word, and the MCM field of the modals.

Check Header: The 50 Series processors use a 16-byte save area in memory to contain information about the system. This check header is located in segment 4 (the interrupt segment). Table 10-12 shows the format of the check block.

Table 10-12
Check Header Format

Offset	Contents
0 to 1	PBH, PBL
2 to 3	KEYSH, KEYSL (modals)
4 to 7	Software code (possibly a JST instruction)

Check Vectors: Segment 0 locations starting at '200 can contain the four check vectors. Check vectors are 16-bit indirect pointers with the format shown in Figure 3-4. The 50 Series processors use these vectors in check handling only when PXM mode is disabled.

Diagnostic Status Words: The 50 Series processors also use a group of 32-bit registers called the diagnostic status words (DSWs). The check handler uses the DSWs as a source of information about the system as it was when the check occurred. The format of the each DSW register is shown in the following tables. (For the DSWPARITYs and DSWSTATs of the earlier systems listed on page 1-1, see Appendix B.)

<u>DSW Register</u>	<u>System</u>	<u>Table</u>
DSWPARTY	6350	10-13
DSWPARTY2	6350	10-14
DSWPARTY	9750 to 9955 II	10-15
DSWPARTY	2350 to 2755, 9650, and 9655	10-16
DSWSTAT	6350	10-17
DSWSTAT	9750 to 9955 II	10-18
DSWSTAT	2350 to 2755, 9650, and 9655	10-19
DSWRMA	All 50 Series	10-20
DSWPB	All 50 Series	10-21

Table 10-13
Format of DSWPARITY Register for the 6350

Bits	Name	Description
1	I/O Parity Error	If 1, the control store reported an I/O parity error. Sets bits 2 to 7 to specify the location of the error: Bit 2: BPD high side, left byte Bit 3: BPD high side, right byte Bit 4: If DSWPARITY bit 11 set to 1, BPA high left byte If DSWSTAT bit 15 set to 1, BPA high left byte or BPD low left byte Otherwise, BPD low left byte Bit 5: If DSWPARITY bit 11 set to 1, BPA high right byte If DSWSTAT bit 15 set to 1, BPA high right byte or BPD low right byte Otherwise, BPD low right byte Bit 6: BPA, detected by PIOS board Bit 7: BPD, detected by PIOS board
2 to 7	I/O Parity Error Code	Specifies the location of the I/O parity error. See bit 1 above for details.
8 to 10	ROC Parity Error Code	If DSWSTAT bit 12 is set to 1, these three bits contain the ROC parity error code: 000: FROCPE1 001: FROCPE2 010: FROCPE3 011: FROCPE4 100: FROCPE5 101: FROCPE6 110: FROCPE7 111: FROCPE8
11	Interrupt Parity Error	If 1, the parity error occurred in an interrupt.
12	Decode Net High Parity Error	If 1, the control store reported a parity error in the decode net's high side.
13	Decode Net Low Parity Error	If 1, the control store reported a parity error in the decode net's low side.
14 to 16	E Unit Parity Error Code	Describes the E unit parity error status: 000: no error 001: no error 010: BAH parity error 011: BAL parity error 100: BAE parity error 101: BEH parity error 110: BEL parity error 111: BBE parity error

Table 10-13 (continued)
Format of DSWPARITY Register for the 6350

Bits	Name	Description
17	Lost Memory Error	If 1, the memory control unit reported a lost error.
18	Memory Address Shift Control	Specifies the address size of a slot in the memory backplane: 0: 8-megabyte slot decode 1: 16-megabyte slot decode
19 to 26	Memory Array Error	A 1 in any of these bits specifies the memory array that reported the error: Bit 19: Memory array number 1 Bit 20: Memory array number 2 Bit 21: Memory array number 3 Bit 22: Memory array number 4 Bit 23: Memory array number 5 Bit 24: Memory array number 6 Bit 25: Memory array number 7 Bit 26: Memory array number 8
27 to 32	Memory Control Unit-Reported Errors	A 1 in any of these bits specifies the error reported by the memory control unit: Bit 27: BB parity error Bit 28: BD parity error Bit 29: BIP in parity error Bit 30: BIP out parity error Bit 31: memory time out error Bit 32: CIT error

Table 10-14
Format of DSWPARITY2 Register for the 6350

Bits	Name	Description
1 to 4	IS Unit-Reported Memory Errors	A 1 in any of these bits specifies the error reported by the IS unit: Bit 1: BDH left parity error Bit 2: BDH right parity error Bit 3: BDL left parity error Bit 4: BDL right parity error
5 to 8	---	Currently unused.
9	Fatal Cache Parity Error	If 1, the control store reported a fatal cache parity error.
10	Branch Cache Recoverable Error	If 1, the control store reported a branch cache recoverable error.
11 to 22	IS Unit-Reported Cache Parity Errors	A 1 in any of these bits specifies the cache parity error reported by the IS unit: Bit 11: cache data parity error on Element B even data low byte Bit 12: cache data parity error on Element B odd data low byte Bit 13: cache data parity error on Element A even data low byte Bit 14: cache data parity error on Element A odd data low byte Bit 15: cache index parity error on Element B low byte Bit 16: cache index parity error on Element B high byte Bit 17: cache data parity error on Element A even data high byte Bit 18: cache data parity error on Element A odd data high byte Bit 19: cache data parity error on Element B even data high byte Bit 20: cache data parity error on Element B odd data high byte Bit 21: cache index parity error on Element A high byte Bit 22: cache index parity error on Element A low byte

Table 10-14 (continued)
Format of DSWPARITY2 Register for the 6350

Bits	Name	Description
23 to 32	IS Unit-Reported STLB Parity Errors	<p>A 1 in any of these bits specifies the STLB parity error reported by the IS unit:</p> <p>Bit 23: STLB parity error on Element B physical address low byte</p> <p>Bit 24: STLB parity error on Element A physical address low byte</p> <p>Bit 25: STLB parity error on Element B access bits</p> <p>Bit 26: STLB parity error on Element B process ID</p> <p>Bit 27: STLB parity error on Element B virtual address tag</p> <p>Bit 28: STLB parity error on Element A physical address high byte</p> <p>Bit 29: STLB parity error on Element B physical address high byte</p> <p>Bit 30: STLB parity error on Element A access bits</p> <p>Bit 31: STLB parity error on Element A process ID</p> <p>Bit 32: STLB parity error on Element A virtual address tag</p>

Table 10-15
Format of DSWPARITY Register for the 9750 to 9955 II

Bits	Name	Description
1	ROCPER	If 1, the control store detected an ROC parity error. Sets bits 3 to 8 of DSWPARITY to reflect the state of the parity error: Bits 3 to 5: encoding of ROC parity error bits 1 to 8 Bit 6: logical OR of ROC parity bits 1 to 8 Bit 7: ROC parity error bit 9 Bit 8: 0
2	IOPER	If 1, the control store detected an I/O parity error. Sets bits 3 to 8 of DSWPARITY to reflect the state of the I/O parity error: Bit 3: error is in left byte of either BPA or BPD Bit 4: error is in right byte of either BPA or BPD Bit 5: CPU detected a parity error on BPD Bit 6: CPU detected a parity error on BPA Bit 7: controller detected a parity error on BPD Bit 8: controller detected a parity error on BPA
3 to 8	Parity Error Code	Specifies information about the ROC or I/O parity error that occurred. See bits 1 and 2 above for specifics.
9	---	Currently unused.
10	BBH Left Byte Parity Error	If 1, the E1 board detected a parity error on BBH, left byte.
11	BBH right Byte Parity Error	If 1, the E1 board detected a parity error on BBH, right byte.
12	BEL Left Byte Parity Error	If 1, the E1 board detected a parity error on BEL, left byte.
13	BEL Right Byte Parity Error	If 1, the E1 board detected a parity error on BEL, right byte.
14	BAH Parity Error	If 1, the E1 board detected a parity error on BAH.
15	BAL Parity Error	If 1, the E1 board detected a parity error on BAL.
16	BAE Parity Error	If 1, the E1 board detected a parity error on BAE.

Table 10-15 (continued)
Format of DSWPARITY Register for the 9750 to 9955 II

Bits	Name	Description
17	ED Parity Error	If 1, the memory control unit detected a parity error on ED. Sets bits 20 to 23 to reflect the error's location. Bit 20: EDH, left byte Bit 21: EDH, right byte Bit 22: EDL, left byte Bit 23: EDL, right byte
18	Memory Data Parity Error	If 1, the memory control unit detected a latched memory data error. Sets bits 20 to 23 of DSWPARITY to reflect the error's location: Bit 20: LMDH, left byte Bit 21: LMDH, right byte Bit 22: LMDL, left byte Bit 23: LMDL, right byte
19	Memory Address Parity Error	If 1, the memory control unit detected a latched memory address error. Sets bits 20 to 23 of DSWPARITY to reflect the error's location: Bit 20: MCADDR, high byte Bit 21: MCADDR, low byte Bit 22: MCADDR, extended byte Bit 23: unused
20 to 23	Parity Error Code	Specifies information about the ED, memory data, or memory address parity error that occurred. See bits 17, 18, and 19 above for specifics.
24	MC ECCU Error	If 1, the memory control unit detected an ECC uncorrectable error.
25	I Unit Error	If 1, the I unit detected an error. Bits 26 to 28 describe the error.
26 to 28	I Unit Error Code	000: no error 001: currently unused 010: currently unused 011: decode net, right byte 100: decode net, left byte 101: base register file high 110: base register file low 111: index register file

Table 10-15 (continued)
Format of DSWPARITY Register for the 9750 to 9955 II

Bits	Name	Description
29	F or S Unit Error*	If 1, the F or S unit detected an error. Bits 30 to 32 describe the error.
30 to 32	F or S Unit Error Bits*	For the 9955 and 9955 II only: 000: no error 001: LPID out of STLB in error 010: LBPA out of STLB in error 011: LBVA out of STLB in error 100: ARR out of STLB in error 101: cache index 110: cache data high side 111: cache data low side For the 9750 to 9950 only: 000: PID or STLB control bits 001: LBPA out of STLB in error 010: cache index, right 16 bits 011: cache index, left 16 bits 100: cache data high side 101: cache data low side 110: LBVA out of STLB in error 111: branch cache parity error

*The F board for the 9955 and 9955 II; the S unit for the 9750 to 9950.

Table 10-16

Format of DSWPARITY Register
For the 2350 to 2755, 9650, and 9655*

Bits	Name	Description
1	Wide Word Mode	If 1, then the last memory operation performed was wide word.
2	Interleaved	If 1, then the last memory operation error.
3	STLB	If 1, an STLB parity error occurred.
4	Cache Index	If 1, a cache index parity error occurred.
5	Cache Data Even	If 1, a cache data even parity error occurred.
6	Cache Data Odd	If 1, a cache data odd parity error occurred.
7	BMD B Board	If 1, a BMD backplane parity error occurred.
8	BPD B Board	If 1, a BPD backplane parity error occurred.
9	RFH Left Byte	If 1, an RFH left byte parity error occurred.
10	RFH Right Byte	If 1, an RFH right byte parity error occurred.
11	RFL Left Byte	If 1, an RFL left byte parity error occurred.
12	RFL Right Byte	If 1, an RFL right byte parity error occurred.
13	RFH Late	If 1, an RFH parity error occurred during the late cycle.
14	RFL Late	If 1, an RFL parity error occurred during the late cycle.
15	BMD or BPD A-Board	If 1, a BMD or BPD parity error occurred as read onto the A-board.
16	BMA or BPA A-Board	If 1, a BMA or BPA parity error occurred as read onto the A-board.
17	RCM**	If 1, an RCM parity error occurred.
18	BMA**	If 1, a BMA parity error occurred.
19	BPA**	If 1, a BMA parity error occurred.
20 to 32	-----	Reserved for future use.

* All parity errors mentioned in this table are single bit.

** Unused by the 2350 to 2655, 9650, and 9655.

Table 10-17
Format of DSWSTAT Register for the 6350

Bits	Name	Description
1	Check Immediate	If 1, the check was taken immediately.
2	Machine Check	If 1, a machine check occurred.
3	Memory Parity	If 1, a memory parity error occurred.
4	Missing Memory Module	If 1, a missing memory module caused the check.
5	E Unit	If 1, the E unit reported a parity error.
6	IS Unit	If 1, the IS unit reported a parity error.
7	CS Unit	If 1, the control store board reported a parity error.
8	MC Unit	If 1, the memory controller unit reported a parity error.
9	ECCU	If bits 3 and 9 are both 1, the memory parity error was ECC uncorrectable.
10	ECCC	If bits 3 and 10 are both 1, the memory parity error was ECC correctable.
11	---	Currently unused.
12	RCM Parity	If 1, an RCM parity error was detected by the control store board.
13 to 14	RPBU	Specifies the RP backup count at the time of the error.
15	DMx Operation	If 1, a DMx transfer was in progress when the error occurred.
16	I/O Operation	If 1, an I/O operation was in progress when the error occurred.
17 to 23	ECC Syndrome Bits	A memory parity error occurred as encoded in these bits; see Table 10-27.
24	Memory Module Number	If a memory error occurred, this bit identifies the interleaved memory module with the error (bit 15 of address).
25	RMA Invalid	If 1, the contents of DSWRMA are invalid.
26	Recoverable Parity Error	If 1, a recoverable parity error occurred in the cache, STLB, or branch cache.
27	Hard Parity Error	If bit 26 is 1 and this bit is also 1, a recoverable hard parity error occurred. There is a permanent error in hardware.
28	---	Currently unused.
29	Internal Error	If 1, the microcode detected an internal error. DSWRMA has a code that indicates the failing microcode algorithm.
30	Dual Config	If 1, a dual processor had both CPUs running at the time of the error.
31	Slave Error	If bit 30 is 1 and this bit is also 1, the slave processor reported the error.
32	Memory Bus Code	Specifies the memory bus with the error (bit 14 of the address).

Table 10-18
Format of DSWSTAT Register for the 9750 to 9955 II

Bits	Name	Description
1	Check Immediate	If 1, the check was taken immediately.
2	Machine Check	If 1, a machine check occurred.
3	Memory Parity	If 1, a memory parity error caused the check.
4	Missing Memory Module	If 1, a missing memory module caused the check.
5	E1 Unit	If 1, the E1 board reported a parity error.
6	F or S Unit*	If 1, the F or S unit reported a parity error.
7	I Unit	If 1, the I unit reported a parity error.
8	MC Unit	If 1, the memory controller unit reported a parity error.
9	ECCU	If bits 3 and 9 are both 1, the memory parity error was EOC uncorrectable.
10	ECCC	If bits 3 and 10 are both 1, the memory parity error was EOC correctable.
11	CS Unit	If 1, the control store board reported a parity error.
12	RCM Parity	If 1, an RCM parity error was detected by the control store board.
13 to 14	RPBU	Specifies the RP backup count at the time of the error.
15	DMx Operation	If 1, a DMx transfer was in progress when the error occurred.
16	I/O Operation	If 1, an I/O operation was in progress when the error occurred.
17 to 23	EOC Syndrome Bits	If a memory parity error occurred, these bits describe the error. See Table 10-28.
24	Memory Module Number	If a memory error occurred, this bit identifies the interleaved memory module that contained the error (bit 15 of address at time of error).
25	RMA Invalid	If 1, the contents of DSWRMA are invalid.
26	Recoverable Machine Check	For the 9955 and 9955 II: If 1, a recoverable machine check occurred. For the 9750 to 9950, this bit is currently unused.
27 to 32	---	Currently unused.

* The F board for the 9955 and 9955 II;
the S unit for the 9750 to 9950.

Table 10-12

Format of DSWSTAT Register
For the 2350 to 2755, 9650, and 9655

Bits	Name	Description
1	Check Immediate	If 1, the check was taken immediately.
2	Machine Check	If 1, a machine check occurred.
3	Memory Parity	If 1, a memory parity error caused the check.
4	Missing Memory Module	If 1, a missing memory module error caused the check.
5 to 7	Machine Check Code	The hardware detected the cause of the trap as follows: 000: none 001: BPD parity 010: BMD parity 011: cache data 100: BPA parity 101: STLB parity 110: BMA parity 111: A-board parity
8	NOT RCM	If 1, the error did not occur in the control unit memory.
9	ECCU	If bits 3 and 9 are both 1, the memory parity error was ECC uncorrectable.
10	ECCC	If bits 3 and 10 are both 1, the memory parity error was ECC correctable.
11	BUP INV	If 1, the RP backup count in bits 12 to 14 is invalid.
12 to 14	RP BAK	Specifies the RP backup count, which is the amount DSWPB was incremented in the current instruction.
15	DMx Operation	If 1, a DMx transfer was in progress when the error occurred.
16	I/O Operation	If 1, an I/O operation was in progress when the error occurred.
17 to 21	ECC Syndrome Bits	If a memory parity error occurred, these bits describe the error. Used with the overall parity bit in Table 10-29.
22	Overall Parity	The value of this bit indicates the overall parity. Used with the ECC syndrome bits in Table 10-29.
23	---	Currently unused.
24	Memory Module Number	The low order address bit of the module indicated in error.
25	RMA Invalid	If 1, the contents of DSWRMA are invalid.
26 to 32	---	Currently unused.

Table 10-20
Format of the DSWRMA Register for All 50 Series Systems

Bits	Description
1 to 32	<p>This is the DSW memory address register. Its validity and format are presented below.</p> <p><u>Validity for All 50 Series:</u></p> <p>Valid if an ECCC, EOCU, recoverable machine check (6350, 9955, and 9955 II only), or missing memory module check occurred.</p> <p>Invalid if any other checks occurred, or if no check occurred.</p> <p><u>Format If Valid:</u></p> <p>Note: In the event of multiple checks, DSWRMA is the RMA of the missing memory module check if there is one. If not, it is the RMA of the machine or ECC-uncorrected check (they are mutually exclusive) if there is one. If not, it is the RMA of the ECC-corrected check.</p> <p><u>For 6350, 9955, and 9955 II Only:</u></p> <p>For a recoverable machine check, DSWRMA contains a 32-bit virtual address at the time of an STLB parity error; when a soft cache parity error is reported, DSWRMA is cleared to 0; for a hard cache parity error (6350 only), DSWRMA contains the cache index address that corresponds to the cache data with the error. When an internal error is reported (6350 only), DSWRMA contains a code for the microcode error. For all other checks, DSWRMA bits 4 to 16 contain bits 1 to 13 of a physical address at the time of the error. Bits 17 to 32 of DSWRMA contain 0.</p> <p><u>For 9750 to 9950:</u></p> <p>DSWRMA bits 4 to 16 contain bits 1 to 13 of a physical address at the time of the error. Bits 17 to 32 contain 0.</p> <p><u>For Rest of 50 Series:</u></p> <p>DSWRMA contains a 32-bit virtual address at the time of the error.</p>

Table 10-21
Format of the DSWPB Register for All 50 Series Systems

Bits	Description
1 to 32	Always valid. The format is the extended program counter (ring, segment, offset). In the event of multiple checks, DSWPB is the program counter of the missing memory module check, if there is one. If not, it is the program counter of the machine or ECC-uncorrected check (which are mutually exclusive) if there is one. If not, it is the program counter of the ECC-corrected check.

Each time the processor performs a check (except for power failure), it sets particular register file locations to reflect the contents of the DSW, as shown in Table 10-22.

Table 10-22
DSW Value After Checks

RF Location*	Contents
'24	DSWPARITY2**
'27	DSWPARITY***
'34	DSWRMA
'35	DSWSTAT
'36	DSWPB

* These are absolute locations in the register file.

** DSWPARITY2 is used only on the 6350.

*** For the earlier systems listed on page 1-1, DSWPARITY is used only on the 750 and 850 as presented in Appendix B.

MCM Field: The 50 Series processors use the MCM field (bits 15 and 16) of the modals to determine what kind of check reporting to do. Table 10-23 shows the possible modes of reporting.

Table 10-23
Modes of Check Reporting

Modals Bits 15 and 16	Reporting Mode
00	No reporting.
01	Report uncorrected memory errors (ECCU) only.
10	Report fatal (ECCU, machine checks, and missing memory module) errors only. (This state is called quiet mode.)
11	Report all errors, including ECCC and, for the 6350, 9955, and 9955 II, recoverable machine checks. (This state is called noisy mode.)

Check Handler Operation

As with faults, the type of check service provided depends on whether the PXM is enabled or not when the check occurs. If the PXM is disabled, the processor sets the MCM field in the modals to 0, then jumps indirectly (JST) through the appropriate check vector to the check routine.

If the PXM is enabled, the processor:

1. Sets up a check header.
2. Inhibits the machine.
3. Switches to 64V mode.
4. Sets the MCM field to 0 (2 if ECCC or recoverable machine check).
5. Transfers control to the check handler.

The software must clear the DSW after each check. This ensures that the processor does not use old data when servicing future checks.

The DSW is large enough to contain data about one of each type of check before the handler takes control. However, the values of RMA and PB for the last check only are saved.

To determine which check stored RMA and PB, use DSWSTAT to determine which checks have occurred:

- If a missing memory module check, machine check, or ECCU memory check occurred, DSWRMA and DSWPB reflect values stored by that check. These three checks are mutually exclusive, and are guaranteed to be the most recent check that occurred.
- If any other check occurred, DSWRMA and DSWPB reflect values stored by the ECCC check or recoverable machine check that occurred most recently.

Table 10-24 summarizes some information about each check.

Table 10-24
Types of Checks

Type of Check	Header Loc*	Handler Loc*	Effect on DSW
Power failure	'200	'204	Does not set DSW.
Environment**	'200	'204	Does not set DSW.
Memory parity	'270	'274	Sets DSW.
Machine check	'300	'304	Sets DSW.
Missing memory module	'310	'314	Sets DSW.
Recoverable machine check***	'320	'324	Sets DSW.

* These are locations in Segment 4.

** Unavailable for the earlier processors listed on page 1-1.

*** For the 6350, 9955, and 9955 II only.

Check Trap

Some checks cause a microcode trap when they occur. When this happens, the action taken depends on the type of microcode that was trapped. Table 10-25 shows the checks that can cause traps and the actions that occur.

The first and second categories listed in Table 10-25 always leave the I/O bus clean.

Table 10-25
Check-produced Traps and Their Actions*

Event	Actions
Missing Memory Module, ECC Uncorrectable, or Machine Check during I/O (DMx, PIO, interrupt processing, excepting machine check for RCM parity)	The error is ignored until all current requests for DMx and I/O are processed, and then the check is taken immediately.
ECC Correctable Error (not during I/O)	Action is deferred until the next fetch cycle, and then a check is taken.
Recoverable machine check (6350, 9955, and 9955 II only)	For an STLB parity error, saves the RS contents in RSSAV and forces an STLB miss. If the parity error has gone away, restores RS contents, causes a fetch cycle trap, and then a check is taken.
Power Failure; Environment	Action is deferred until the next fetch cycle, and then a check is taken.
All other checks	Software check occurs immediately.

* For the actions that can occur with the earlier processors listed on page 1-1, see Appendix B.

TRAPS

Traps are breaks in microcode execution. When a trap occurs, the processor takes the current microcode location where the trap occurred and goes to the predetermined microcode location that handles traps. The processor handles the trap, then retries the microcode location where the trap originally occurred.

Traps are separated into two groups. G1 traps occur during references to parts of memory. G2 traps are hardware related and encompass several subgroups. Table 10-26 lists the traps in both groups and the further subgroups.

The traps are listed in order of priority, from highest to lowest. G2 traps always have higher priority than G1 traps do. Within the G2 group, missing memory module traps have the highest priority. (See Appendix B for the priority of the G2 traps for memory increment interrupts, supported only on the earlier processors listed on page 1-1.)

Table 10-26
Types of Traps and Their Priorities

Type	Individual Trap	Causes and Actions
G1	32-bit or 16-bit read address trap	If a memory reference instruction forms an EA between 0 to '7 (V mode) or 0 to '37 (S and R modes), the addressed location is in the current user register file, not memory. When such an address is calculated, this trap aborts the memory read and loads a cache entry with the contents of the addressed register. The cache is marked invalid but its Use Once bit is set to 1 so that a cache hit occurs when the microstep is retried. A cache miss occurs on this entry's next reference.
	STLB miss	This trap aborts the step. The STLB miss translates the virtual address to a physical one, then puts the translation into the STLB. The step is retried after the translation is loaded into the STLB.
	Access violation	A procedure tries to reference a memory location for which it has insufficient access rights. This trap causes an access violation fault.
	Page modified trap	This trap occurs in each step that writes into a physical page whose modified bit (in the page's STLB entry) contains 0. This trap sets the modified bit to 1 so that future writes to this page do not cause other traps.
G2	Missing memory module	If no memory board responds to a memory read or write request, this trap occurs. Actions taken as a result of this trap depend on the operating system.
	Machine check	Indicates a parity error, EOCU, or (6350 only) an internal (microcode) error. The parity error type is in DSWPARITY. See Checks in this chapter for more information. This is a fatal trap.
	Write address trap	Specifying an address within the range 0 to '7 (V mode) or 0 to '37 (S and R modes) as a write address causes this trap which aborts the write to memory but otherwise allows the operation to complete.

Table 10-26 (continued)
Types of Traps and Their Priorities

Type	Individual Trap	Causes and Actions
G2 (cont)	Integer exception	The current instruction caused an integer exception. This trap causes an integer exception fault.
	Branch cache problem	A branch cache hit occurs during execution of something other than a branch instruction.
	DMx requests	If a controller wants to request a DMx transfer, this trap transfers control to the DMx microcode.
	Fetch cycle traps:	Allows the processor to perform several steps between microsteps.
	-- CPU timer overflow	The microsecond timer overflows. This trap increments the contents of TIMER by 1. If the incremented value of TIMER does not overflow, execution continues. If it does overflow, this trap sets the process abort flag in the process' PCB to 1.
	-- Diagnostic processor interrupts	The diagnostic processor sends a command to the processor. The microcode reads the command and decodes it. Invalid for earlier processors 750 and 850.
	-- ECCC	Error correcting codes on the memory boards note when single bit errors in MOS memory occur. This trap notes the address where such an error occurred and the value of that address' syndrome bits. The syndrome bits show which bit in that location is in error. See Tables 10-27 to 10-29 for information about syndrome bit values for single bit errors.
	-- External interrupts	Point where a device requested service. This trap causes an external interrupt. (See <u>Interrupts</u> , earlier in this chapter, for more information.)

Table 10-26 (continued)
Types of Traps and Their Priorities

Type	Individual Trap	Causes and Actions
G2 (Cont)	-- Cache parity error*	A cache parity error occurred. For the 9955 and 9955 II, this error is reported at the next PIC interrupt to DSWPARITY, DSWSTAT, DSWRMA, and DGR12. This error then goes to the check handler. (See <u>Checks</u> in this chapter.) When this error occurs on a 6350, a cache portion is scanned every PIC for a hard parity error; if one is not found, the actions taken are as for an STLB parity error except that a cache parity error clears DSWRMA.
	-- Program interval timer	If the timer is enabled, it causes an interrupt. The timer places a vector on the address bus for PRIMOS.
	-- STLB parity error*	Caused by an STLB recoverable parity error trap. The virtual address at the time of error is loaded in DSWRMA. Sets the REOIV and updates DSWPARITY, DSWSTAT, and (9955 and 9955 II only) DGR12. Control goes to the check handler.
	-- Hard parity error**	For STLB: detected and treated like a soft (recoverable) STLB parity error, but the Hard Parity Error bit is set in DSWSTAT and the bad STLB location is mapped out to limit performance degradation. For cache: can be detected during a scan of the cache upon a cache parity error; maps out the bad cache location and sets DSWSTAT's Hard Parity Error bit.
	-- End-of-instruction trap	A parity error occurred on an I/O transfer. Not used by the 6350 and 9750 to 9955 II.
	-- Power failure	AC power failed. This trap causes a power failure check. (See <u>Checks</u> .)
	Restricted instruction trap	This trap causes a fault when a process tries to execute a restricted instruction in a ring other than Ring 0.

* For the 6350, 9955, and 9955 II only.

** For the 6350 only.

Read Address Trap

If the effective address calculated by a memory reference instruction is within the range 0 to '7 (V mode) or 0 to '37 (S and R modes), inclusive, the addressed location is in the current user register file, not in memory. When such an address is calculated, this trap aborts the memory read and loads a cache cell with the contents of the addressed register. The cache is marked invalid but the cache's use once bit is set to 1 so that a cache hit occurs when the microstep is retried. The cache miss occurs on the next reference to this cache cell.

STLB Miss

When an STLB miss occurs, this trap aborts the step. The STLB miss translates the virtual address to a physical one, then puts the translation into the STLB. The step is retried after the translation is loaded into the STLB.

Access Violation

If one procedure tries to call another and an access violation occurs, this trap causes an access violation fault.

Page Modified

This trap occurs during each step that writes into a physical page whose modified bit (in the page's STLB entry) contains a 0. This trap sets the modified bit to 1 to indicate the presence of information that must be saved.

Missing Memory Module

If no memory board responds to a memory read or write request, this trap occurs. A missing memory module check alerts the operating system to this trap's occurrence; resulting actions depend on the operating system.

Error Correcting Code

Error correcting codes on the memory boards note when single bit errors in MOS memory occur. This trap notes the address where such an error occurred and the value of that address' syndrome bits that tell which bit in that location is in error. Tables 10-27 to 10-29 show the values of the syndrome bits and the single bit errors they indicate.

Table 10-27
Syndrome Bits for the 6350

Synd Bits 1234567	Bit in Error	Synd Bits 1234567	Bit in Error
0000000	No Error	1100100	Word bit 13
1000000	Check bit 1	0110100	Word bit 14
0100000	Check bit 2	0101100	Word bit 15
0010000	Check bit 3	0011100	Word bit 16
0001000	Check bit 4	1010010	Word bit 17
0000100	Check bit 5	1111010	Word bit 18
0000010	Check bit 6	1001010	Word bit 19
0000001	Check bit 7	0011010	Word bit 20
1110011	Word bit 01	1000110	Word bit 21
1011011	Word bit 02	0010110	Word bit 22
1101011	Word bit 03	0001110	Word bit 23
0111011	Word bit 04	0111110	Word bit 24
1100111	Word bit 05	1010001	Word bit 25
0110111	Word bit 06	1111001	Word bit 26
0101111	Word bit 07	1001001	Word bit 27
0011111	Word bit 08	0011001	Word bit 28
1110000	Word bit 09	1000101	Word bit 29
1011000	Word bit 10	0010101	Word bit 30
1101000	Word bit 11	0001101	Word bit 31
0111000	Word bit 12	0111101	Word bit 32

Note to Table 10-27

A multiple error has occurred when the syndrome bits have a pattern not shown above.

Table 10-28
Syndrome Bits for the 9750 to 9955 II

Check Bits 6543210	Bit in Error	Check Bits 6543210	Bit in Error
0000000	No error	0101100	Word bit 13
0000001	Check bit 0	1101101	Word bit 14
0000010	Check bit 1	1101110	Word bit 15
0000100	Check bit 2	0101111	Word bit 16
0001000	Check bit 3	1110000	Word bit 17
0010000	Check bit 4	0110001	Word bit 18
0100000	Check bit 5	0110010	Word bit 19
1000000	Check bit 6	1110011	Word bit 20
0000111	Word bit 01	0110100	Word bit 21
1100001	Word bit 02	1110101	Word bit 22
1100010	Word bit 03	1110110	Word bit 23
0100011	Word bit 04	0110111	Word bit 24
1100100	Word bit 05	0111000	Word bit 25
0100101	Word bit 06	1111001	Word bit 26
0100110	Word bit 07	1111010	Word bit 27
1100111	Word bit 08	0111011	Word bit 28
1101000	Word bit 09	1111100	Word bit 29
0101001	Word bit 10	0111101	Word bit 30
0101010	Word bit 11	0111110	Word bit 31
1101011	Word bit 12	1111111	Word bit 32

Note to Table 10-28

A multiple error has occurred when the syndrome bits have a pattern not shown above.

Table 10-29
Syndrome Bits for the Rest of the 50 Series

Check Bits 123456	Bit in Error	Check Bits 123456	Bit in Error
00000X	Multiple bits	10000X	Multiple bits
00001X	Multiple bits	100011	Word bit 07
00010X	Multiple bits	10010X	Multiple bits
000111	Word bit 15	100111	Word bit 03
00100X	Multiple bits	10100X	Multiple bits
001011	Word bit 14	101011	Word bit 02
001101	Word bit 13	101101	Word bit 01
001111	Word bit 09	101111	Check bit 2
01000X	Multiple bits	110001	Word bit 08
01001X	Multiple bits	110011	Word bit 06
01010X	Multiple bits	110101	Word bit 05
010111	Word bit 12	110111	Check bit 5
011001	Word bit 16	111001	Word bit 04
011011	Word bit 11	111011	Check bit 4
011101	Word bit 10	111101	Check bit 3
011111	Right parity/ check bit 1	111111	Overall parity
		111110	No error

Notes to Table 10-29

X means undefined.

In the 2350 to 2755, 9650, and 9655 DSWSTAT, bit 6 of this table is not listed as the sixth ECC syndrome bit but is called the overall parity bit.

Machine Check

This trap, like that for missing memory module, indicates a serious problem with the system. It may indicate faulty components, noise, or a timing problem. For the 6350, it can also indicate problems in the microcode. This is a fatal trap.

Write Address Trap

Specifying an address within the range 0 to '7 (V mode) or 0 to '37 (S and R modes) as a write address causes this trap. This trap aborts the write to memory but otherwise allows the operation to complete.

DMx

If a controller wants to request a DMx transfer, this trap transfers control to the DMx microcode.

Fetch Cycle Traps

Fetch cycle traps occur only at the end of the first microstep of a Prime assembly language instruction. They are caused by a program interval timer overflow, external interrupts, and power failures.

These traps occur only after the first step of an assembly language instruction has completed. This guarantees that the previous assembly language instruction has completed execution.

Restricted Instruction

This trap causes a fault when a process tries to execute a restricted instruction in a ring other than Ring 0.

Summary of Software Breaks Caused by Traps

As mentioned above, some of the traps listed in Table 10-26 cause software breaks. Table 10-30 shows which traps cause additional breaks and the types of breaks that can occur.

Table 10-30
Software Breaks Caused By Traps

Traps	Additional Software Break
Missing memory module; ECCU, machine check, and other parity errors	No additional break occurs. These traps are reported to the operating system via a check; the operating system takes an appropriate action.
External interrupt, memory increment interrupt, program interval timer interrupt	Interrupt occurs.
Integer exception, access violation, restrict mode violation	Fault occurs.
STLB miss	Page fault, segment fault may occur.
Power failure, ECCC	Check occurs.
All other traps	No additional action occurs.

INTERVAL CLOCK

A 250 Hz interval clock is used for all processors but the earlier ones listed on page 1-1 (see Appendix B). If the interval clock is enabled, a fetch cycle trap occurs when a timing pulse occurs. The fetch cycle trap causes an external interrupt. If interrupts are enabled on the machine, the processor services the interrupt and updates the pointers in the clock process. If interrupts are disabled, the interrupt is ignored.

Table 10-31 lists the instructions that control the interval clock.

Table 10-31
Instructions Affecting the Interval Timer

Mnem*	Name	Modes	Description
TNA '1120	Input to A	S, R	Loads the ID of the controller into A.
TNA '1320	Input to A	S, R	Loads the contents of the interrupt vector into A.
OCP '0020	Output Control Pulse	S, R	Starts the interval timer.
OCP '0120	Clear PIC Interrupt	S, R	Clears the phantom interrupt code interrupt.
OCP '0220	Output Control Pulse	S, R	Stops the interval timer.
OCP '1720	Initialize Interval Timer	S, R	Initializes the interval timer.
OTA '0720	Output from A	S, R	Transfers data from A into the control register.
OTA '1320	Output from A	S, R	Transfers data from A into the interrupt vector.
SKS '0020	Skip on Condition Met	S, R	Skips if the interval timer is not interrupting.

* V and I modes execute EIO instructions with these instructions as effective addresses. See Chapter 11 for more information.

SUMMARY

You have read in this chapter about four kinds of breaks in execution that can occur, and how the 50 Series processors handle them. Traps are breaks in microcode execution. Checks indicate hardware consistency problems; faults indicate software exception conditions. External devices issue interrupts when they desire service. The next chapter, Input/Output, shows how external devices issue interrupts, and how the 50 Series processors handle these requests for service.

11

Input-Output

The previous chapter deals with the various types of breaks that can occur in program execution. The I/O system is closely related to these breaks, since data transfers between the processor and other parts of the system usually include some type of break. Depending on the type of transfers and the controller, the I/O system can perform a wide variety of functions applicable to many situations.

I/O on the 50 Series processors is divided into three types:

- Programmed I/O (PIO)
- Direct memory (DMx)
- Interrupts

These three types of I/O differ in what initiates the action. For PIO, the processor issues a command to a controller, which performs the desired action. For DMx transfers, a controller requests service from the processor, which provides the service on a priority basis. For interrupts, the controller again alerts the processor to a situation that requires the processor's attention. Chapter 10 discusses interrupts and how the processor deals with them. This chapter describes PIO and DMx.

PROGRAMMED I/O

PIO is I/O performed by a program. This means that the instruments used to perform PIO are instructions. These instructions:

- Send control information to a peripheral controller.
- Test controllers for skip conditions.
- Move data between a controller and the CPU.

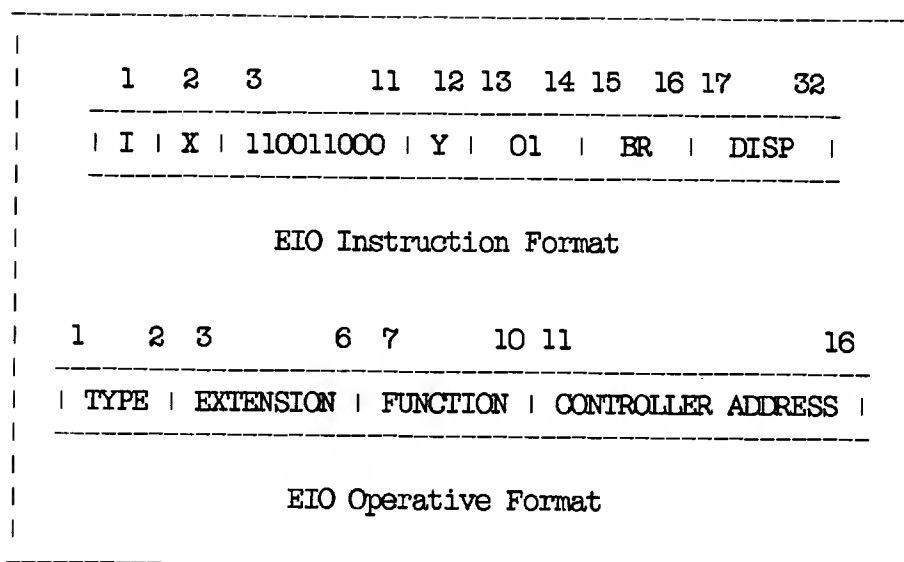
The PIO instructions use one of two formats. In S mode and R mode, four PIO instructions (OCP, SKS, INA, OTA) are available for use. They have the format shown in Figure 11-1.

1	2	3	6	7	10	11	16
<hr/>							
	TYPE		1100		FUNCTION		CONTROLLER ADDRESS
<hr/>							

INA, OCP, OTA, SKS Operative Format
Figure 11-1

For these four instructions, the operative (the part that the processor actually executes to perform the desired action) is the instruction itself. A different arrangement exists for V mode PIO.

In V mode, the processor cannot directly execute INA, OCP, OTA, or SKS. Instead, it must use an EIO instruction, which forms an effective address. The processor executes bits 17 to 32 of this effective address as a PIO instruction. These bits (the operative of the EIO instruction) should specify one of the four PIO instructions described above. The upper drawing in Figure 11-2 shows the format of the EIO instruction; the lower drawing shows the format the processor uses to interpret the EIO operative.



EIO Formats
Figure 11-2

Both the S and R mode PIO operatives and the V mode operative have the same basic format. In both cases, bits 1 to 6 specify the operation that the processor is to perform. Bits 1 and 2 always identify the basic type of operation to be performed, as shown in Table 11-1.

Table 11-1
Basic I/O Operations

Type	Inst	Name
00	OCP	Output control pulse
01	SKS	Skip if condition satisfied
10	INA	Input to A
11	OTA	Output from A

Bits 3 to 6 have different meanings in different modes. These bits are set to 1100 in S and R modes. In V mode, bits 3 to 6 specify an extension to the type field. The processor may use this field to distinguish between controllers that require different types of service, or between different software implementations. This feature has yet to be completely defined.

In all three modes, the function field (bits 7 to 10) specifies one of 16 controller-dependent commands. Each controller defines the function codes that it uses for each of the four basic PIO operations. For example, the controller for one controller might use INA with a function field of 0 to load data into A, INA with a function field of 1 to load a controller ID into A, and INA with a function field of 3 to load status into A.

In all three modes, bits 11 to 16 specify a controller address that identifies a controller and its implementation. Tables 11-2 and 11-3 show these controller addresses for PRIMOS Rev. 21. Several properties of Table 11-2 require emphasis:

- No two controllers may have the same address in a given system. If two controllers do have the same address, the system will malfunction in unpredictable ways.
- General Purpose Interface Boards (GPIBs) have been assigned controller addresses '60 through '67.
- The addresses '70 to '76 are reserved for manufacturing, special systems, field service, and customers to make special assignments on a system by system basis, because neither standard software nor standard controllers use these addresses. For example, magnetic tape controllers have two assigned addresses, '13 and '14. If a customer wanted a third magnetic tape controller, it could be assigned an address in the '70 to '76 range.

In the past, several controller addresses had been assigned to both a special and a standard option before addresses '70 to '76 were reserved for specials. If a conflict occurs when assigning controller addresses during a field upgrade, the conflict should be resolved by moving the special controller address to one in the '70 to '76 range.

Addresses '75 and '76 are reserved for controllers using the T\$GPPI software driver.

- Prime systems have a configuration list in the cabinet which shows all of the controller addresses in that system and where the boards are plugged in.
- You can generally change controller addresses by inserting one or two header dips in controllers.
- Addresses for the ICS1, ICS2, and ICS3 communications controllers are assigned from a pool of addresses. See Table 11-3.
- Some controller boards such as the 2047 and 2382 disk/tape controllers have separate controller addresses assigned to each function.

Table 11-2
Controller Address Assignments

Address	Model	Controller Description
'00	----	Reserved
'01	3000	Paper tape reader
'02	3000	Paper tape punch
'03	3100	URC #1 (unit record controller): line printer, card reader, card punch
'04	3006	System terminal
'05	3100	URC #2: line printer, card reader, card punch
'06	----	Reserved
'07	7040	Primenet node controller (PNC) #1
'10	2034/2036	Communications controller (see Table 11-3)
'11	2034/2036	Communications controller (see Table 11-3)
'12	4300	Floppy disk
'13	4020	Magnetic tape #2
'14	4020	Magnetic tape #1
'15	2034/2036	Communications controller (see Table 11-3)
'16	2034/2036	Communications controller (see Table 11-3)
'17	2034/2036	Communications controller (see Table 11-3)
'20	3006	Control panel, RTC (realtime clock), SOC (system option controller)
'21	----	Reserved
'22	4004/5/6	Disk controller #3
'23	4004/5/6	Disk controller #4
'24	4004/5/6	Disk controller #5
'25	4004/5/6	Disk controller #6
'26	4004/5/6	Disk controller #1
'27	4004/5/6	Disk controller #2
'30	3007	Buffered parallel I/O channel #1
'31	3025	Buffered parallel I/O channel #2
'32	2034/2036	Communications controller (see Table 11-3)
'33	3009/3008	Versatec printer plotter #1
'34	3009/3008	Versatec printer plotter #2
'35	2034/2036	Communications controller (see Table 11-3)
'36	2034/2036	Communications controller (see Table 11-3)
'37	2034/2036	Communications controller (see Table 11-3)
'40	----	Reserved
'41	----	Reserved
'42	----	Reserved
'43	----	Reserved
'44	----	Reserved
'45	----	Disk controller #7
'46	----	Disk controller #8
'47	7040	Primenet node controller (PNC) #2

Table 11-2 (continued)
Controller Address Assignments

Address	Model	Controller Description
'50	2034/2036	Communications controller (see Table 11-3)
'51	2034/2036	Communications controller (see Table 11-3)
'52	2034/2036	Communications controller (see Table 11-3)
'53	2034/2036	Communications controller (see Table 11-3)
'54	2034/2036	Communications controller (see Table 11-3)
'55	5400	Multiple autocall
'56	2034/2036	Communications controller (see Table 11-3)
'57	----	Reserved
'60	7000	General purpose interface board
'61	7000	General purpose interface board
'62	7000	General purpose interface board
'63	7000	General purpose interface board
'64	7000	General purpose interface board
'65	7000	General purpose interface board
'66	7000	General purpose interface board
'67	7000	General purpose interface board
'70	----	Reserved for specials
'71	----	Reserved for specials
'72	----	Reserved for specials
'73	----	Reserved for specials
'74	----	Reserved for specials
'75	----	Reserved for specials using T\$GPPI driver
'76	----	Reserved for specials using T\$GPPI driver
'77	----	I/O bus tester

Table 11-3 specifies the addresses that may be used for the various types of communications controllers at PRIMOS Rev. 21. The names of the controller types in this table are:

AMLC - Asynchronous Multiline Controller
 SMLC - Synchronous Multiline Controller
 HSSMLC - High Speed Synchronous Multiline Controller
 MDLC - Multiple Data Link Controller
 ICS1 - Intelligent Communications Subsystem 1
 ICS2 - Intelligent Communications Subsystem 2
 ICS3 - Intelligent Communications Subsystem 3

Table 11-3
Communications Controller PIO Addresses

Assignable Addresses	Communications Controllers							
	AMLC	SMC	HSSMLC	MDLC	ICS1	ICS2	ICS3	PNC
'07								#1
'10					#3	#1	#1	
'11					#4	#2	#2	
'15	#5*							
'16	#6*							
'17	#7*							
'32	#8*							
'35	#4*							
'36					#1	#3	#3	
'37					#2	#4	#4	
'47								#2
'50			#1	#1				
'51			#2	#2				
'52	#3*							
'53	#2*							
'54	#1*							
'56		#1						

* The ordering of AMLC addresses is important. The ordering of the other communications controllers is recommended.

Controller identification (ID) numbers are bits assigned to each controller type. Two types of controller IDs are defined. The older type is the single INA '11 ID. The newer one uses INAs '11 and '12. Bits 26 to 32 of INA '11 are either '100 or '077 for the two INA ID implementation and are invalid in the single INA '11 ID. This allows a differentiation between the two types of controllers from the software.

Tables 11-4 and 11-5 show the ID numbers that are currently assigned under both conventions. Some controllers implement both INA IDs for compatibility. All new controllers must use the two INA ID method.

Table 11-4
Controller Board Numbers for Controller IDs using INA '11

INA '11 ID Bits 9 and 10				ID Bits
00	01	10	11	11 to 16
*	*	*	*	'00
*	SOC BPIOC1	*	*	'01
*	SOC BPIOC2	*	*	'02
URC 3156	*	URC 2294	*	'03
Option A	SOC ASLC/SSLC	*	*	'04
*	*	*	*	'05 to '07
*	ICS2 5242,	ICS3 5725,	*	'10
*	5720, 5722	5730, 5735	*	
*	*	*	*	'11 to '12
Tape 2295	Tape D/T 2047	Tape 2382	*	'13
(4190)				
Tape 2081	Tape 2081	Tape 2269/70	Tape 2023	'14
and 4020				
*	*	*	*	'15 to '17
*	SOC LFC, PIC,	*	*	'20
	and WDT			
*	*	*	*	'21
Disk 4000	*	*	*	'22
*	*	*	*	'23 to '25
Disk 4004	Disk 4005,6580	R D/T 2047	*	'26
*	*	*	*	'27 to '35
ICS1 5181	*	*	*	'36
*	*	*	*	'37
PRIMAD A to D	*	*	*	'40
*	*	*	*	'41 to '42
Digital Out	*	*	*	'43
*	*	*	*	'44 to '47
MDLC 5602,	*	*	*	'50
5604/5622/5624				
*	*	*	*	'51 to '53
QAMLC 5154,	*	*	*	'54
5274, 5475				
*	*	*	*	'55 to '60
PNC 7041/42	*	*	*	'61
*	*	*	*	'62 to '77

* Means not valid.

Table 11-5

Controller Board Numbers for Controller IDs Using
INA '11 and '12

Controller	INA '11		INA '12				INA '12	
	Long Word Bits		Bits				Bits	
	26	(27 to 32)	1	2	(3 to 8)	9	10	11 to 16
Tape 2382	1	*	0	0	*	0	0	'00 to '13
		'00			'01			'14
Disk 2382	1	*	0	0	*	0	0	'15 to '25
		'00			'01			'26
Disk 6508	1	'00	0	0	'02	0	0	'26
		*			*			'27 to '77

* Means not valid.

PIO Operative Actions

The processor performs the same actions for each identical PIO operative, regardless of the mode of the machine. This means that the INA operation specified by EIO in V mode results in the same actions as does the INA directly executed in S mode and R mode. After performing the operation, however, the processor indicates the success or failure of the operation in different ways, depending on the mode. The descriptions below explain the actions of each operation, as well as how the processor indicates success or failure for each mode.

INA: INA is enabled over BPA. If the specified controller is not ready and does not have controller address '20, the instruction ends. If the controller is ready or has controller address '20, the controller responds ready and data is read over BPD. In V mode, the condition codes reflect success or failure as shown in Table 11-6.

In S and R modes when the controller address is not '20, the processor indicates success by incrementing the contents of the program counter by 1; when the controller address is '20, no increment occurs.

For controller address '20 the data can have bad parity. INAs to controller address '20 ignore the data parity and generate their own correct parity. INAs to controller addresses other than '20, however, do check the data parity and indicate a BPD parity error if the parity is incorrect.

Table 11-6
Effect of EIO on Condition Codes

OC	Meaning
EQ	Successful INA, OTA, or SKS instruction
NE	Unsuccessful INA, OTA, OR SKS; any OCP

OTA: OTA is enabled over BPA. If the specified controller is not ready and does not have controller address '20, the instruction ends. If the controller is ready or has controller address '20, the controller responds ready and data in A is sent over BPD to the controller. In V mode, the condition codes reflect success or failure as shown in Table 11-6.

In S and R modes when the controller address is not '20, the processor indicates success by incrementing the contents of the program counter by 1; when the controller address is '20, no increment occurs.

SKS: SKS is enabled over BPA. If the specified controller is not ready, the instruction ends. If the controller is ready, the processor indicates success in V mode by setting the condition codes, as shown in Table 11-6, regardless of the controller address.

If the controller is ready, the processor indicates success in S and R modes by incrementing the contents of the program counter by 1, regardless of the controller address.

OCP: OCP is enabled over BPA. The specified controller performs the specified command and the instruction ends. OCP never indicates success or failure.

DMX

While PIO operations are suitable to use when only small amounts of data need to be transferred, they are typically not suitable for multiple data transfers. Each time PIO transfers data, the processor must execute several instructions for each 16-bit quantity transferred. This ratio of control instructions to transferred data makes the transfer of blocks of data rather slow. DMx operations allow controllers to access memory directly, rather than by using software. This cuts down on the amount of processor time required to perform the transfer, and allows the transfer to occur without specific software attention.

DMx Transfers

There are several types of DMx transfers:

- DMA, or direct memory access
- DMC, or direct memory channel
- DMT, or direct memory transfer
- DMQ, or direct memory queue

All of these transfers occur in three phases. The request to transfer occurs during the request phase. The CPU receives the transfer address during the address phase. The data is transferred during the data phase.

To make any DMx request, the controller desiring the transfer sends a DMx request to the processor. This request will be serviced when:

- The processor issues a DMx request enable.
- There are no other DMx requests pending from controllers with a higher priority (a lower slot number).

If the request from this controller has the highest priority, the processor recognizes it. The controller sends an address on BPA and control information on the mode lines. The mode lines request the specific type of DMx transfer, which in turn defines how the address line information is to be interpreted. DMx address formation is described at the end of this chapter.

After receiving the control information, the processor strobes the data as appropriate over BPD. The processor sends an end-of-range (EOR) signal, if appropriate, at the end of the block transfer.

The length of time between when a controller requests service and when the processor responds depends on two things:

- How many requests of higher priority are already pending
- What the processor is doing when the controller makes its request

A controller must wait until the processor services all requests of higher priority. This means that the controller with the highest priority in the system can preempt service to any other controller, and may completely occupy the processor if it transfers data at the maximum rate.

Though the processor can pause between instructions or at selected points within instruction execution, it cannot stop immediately each time a request for a transfer occurs. Also, the processor cannot service requests when servicing interrupts or phantom interrupt code.

This means that even the highest priority controller in the system may have to wait less than 7 microseconds if the processor is busy. Once the processor transfers the first 16 bits, however, it transfers the rest of the block as fast as possible. At the maximum speed, the processor cannot process anything else at the same time.

Maximum rates of transfer for DMA and the other DMx transfers are shown in Table 11-7. Rates for the earlier processors listed on page 1-1 are in Appendix B.

Table 11-7
DMx Transfer Rates

Type	Transfer	Maximum Speed		
		6350	9750 to 9955 II	2350 to 2755, 9650, and 9655
DMA	Input	3.0 Mbytes/sec	2.4 Mbytes/sec	2.2 Mbytes/sec
	Output	1.9 Mbytes/sec	2.0 Mbytes/sec	2.3 Mbytes/sec
Extended DMA	Input	2.2 Mbytes/sec	**	**
	Output	1.6 Mbytes/sec	**	**
16-Bit Burst DMA	Input	9.7 Mbytes/sec	9.4 Mbytes/sec	5.1 Mbytes/sec
	Output	6.6 Mbytes/sec	6.0 Mbytes/sec	4.9 Mbytes/sec
32-bit Burst DMA	Input	24.2 Mbytes/sec	**	**
	Output	11.0 Mbytes/sec	**	**
DMC	Input	1.8 Mbytes/sec	1.2 Mbytes/sec*	700 Kbytes/sec
	Output	1.3 Mbytes/sec	1.1 Mbytes/sec*	700 Kbytes/sec
DMT	Input	2.4 Mbytes/sec	2.8 Mbytes/sec*	2.2 Mbytes/sec
	Output	1.7 Mbytes/sec	2.2 Mbytes/sec	2.3 Mbytes/sec
Burst DMT	Input	8.6 Mbytes/sec	**	**
	Output	5.3 Mbytes/sec	**	**
DMQ	Input	1.1 Mbytes/sec	300 Kbytes/sec*	500 Kbytes/sec
	Output	1.1 Mbytes/sec	300 Kbytes/sec*	500 Kbytes/sec

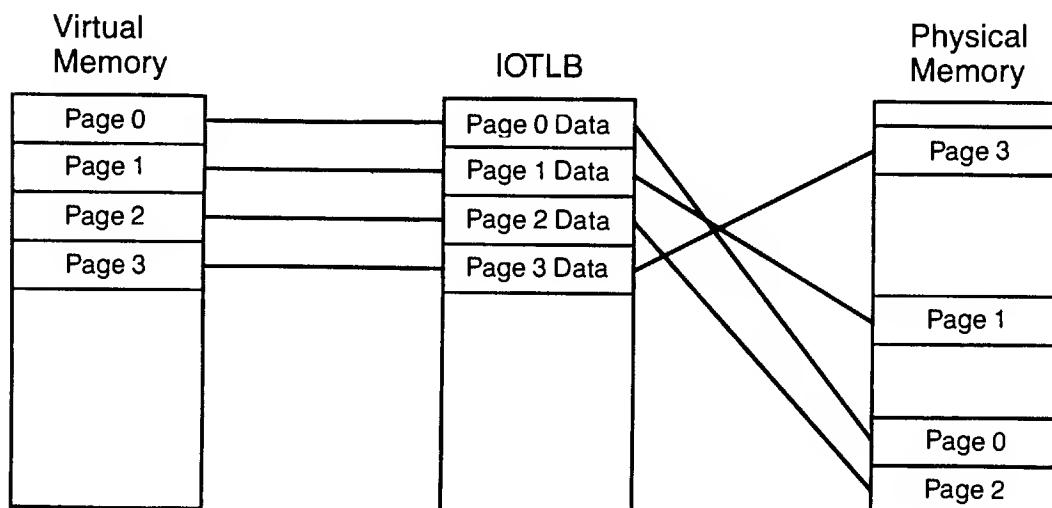
* This is an approximate value.

** Not available for this processor.

Mapped I/O

When a controller specifies the transfer starting address, it can specify a virtual address or a physical one. The processor is using absolute I/O when the address specified is a physical one. When the controller specifies a virtual address, the processor is using mapped I/O.

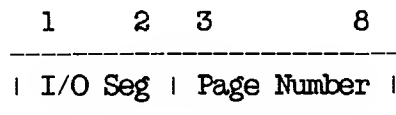
Mapped I/O allows the limited addressing range of DMx transfers to address all of physical memory. It is especially useful when transferring several contiguous pages in virtual memory to physical locations that may not be contiguous. For example, suppose the processor wants to transfer four contiguous pages of data in virtual memory to a controller. As shown in Figure 11-3, mapped I/O allows the system to map the four pages to any four available pages, instead of requiring one four-page block.



Mapped I/O
Figure 11-3

The IOTLB contains the information needed to map the transfer addresses to physical memory locations. The IOTLB, with the STLB, forms the virtual-to-physical address mapping hardware and contains 256 entries for the 6350; there are 128 entries for the 2350 to 2755 and 9650 to 9955 II. (Appendix B discusses the IOTLB of the earlier processors listed on page 1-1.)

An 8-bit address selects each IOTLB entry. This address is composed of the I/O segment number and the page number in that I/O segment as shown in Figure 11-4.



Address Format of IOTLB Entry
Figure 11-4

Each IOTLB entry contains mapping information for one page of the I/O segments as shown in Table 11-8.

Table 11-8
IOTLB Mapping

IOTLB Entry	Corresponding Page in I/O Segments
0 to 63	Segment 0, Pages 0 to 63
64 to 127	Segment 1, Pages 0 to 63 (2350 to 9955 II only)
128 to 191	Segment 2, Pages 0 to 63 (6350 only)
192 to 255	Segment 3, Pages 0 to 63 (6350 only)

The IOTLB allows the I/O address translation during DMx to be done swiftly because information about the translation is always guaranteed to be in the IOTLB. If the processor were to rely on the STLB, an STLB miss could occur and the transfer would fail. Preloading the IOTLB is, therefore, essential before initiating I/O.

The LIOT instruction loads the IOTLB entries with transfer information. This instruction must be used before any transfer occurs so that the processor maps virtual pages to the desired physical ones. (Appendix B discusses LIOT for the earlier processors listed on page 1-1.)

Table 11-9 shows the contents of each IOTLB entry. (Appendix B discusses the IOTLB of the earlier processors listed on page 1-1.)

Table 11-9
IOTLB Entry Format

Field	Number of Bits	Description
Physical page number	16 for 6350 14 for 9955 II 13 for 9750 to 9955 12 for 2350 to 2755, 9650, and 9655	Specifies the physical page address.
Valid bit	1 for all processors	Indicates if this entry contains old data.
MBIO bits	5 for 2755, 9955, and 9955 II 3 for 2350 to 2655 and 6350 to 9950	Specifies the cache leaf to invalidate when writing to memory.

Since the cache of the 2755, 9955, and 9955 II contains 64K bytes, it can contain mapping information for 32 entries of physical memory, each having the same page offset. This is called a 32-leaf cache. MBIO bits allow the information for only the modified entry to be invalidated after a memory write, rather than each of the 32 possible places. Thus, these MBIO bits determine which leaf of the cache to invalidate after a memory write. Five MBIO bits are used for a 32-leaf cache.

The 6350 has a 32K-byte two-set associative cache. This cache can contain mapping information for 16 entries of physical memory, each having the same page offset. Three MBIO bits are used to specify which cache leaf contains an entry that may be invalidated after a memory write. Since each cache access returns two cache index entries, however, the 6350 invalidates only the entry whose physical page address matches that of the IOTLB entry.

The 2350 to 2655 and 9650 to 9950 have a 16K-byte cache containing mapping information for 8 entries of physical memory. Thus, these processors have an 8-leaf cache that requires 3 MBIO bits to specify which cache leaf to invalidate after a memory write.

DMA

DMA is useful for bulk data transfers when speed is important, and can also be operated in burst mode as described in a further section. Of the earlier systems listed on page 1-1, burst mode is used only with the 750 and 850.

Maximum rates of transfer for all forms of DMA are shown in Table 11-7, located earlier in this chapter.

The register file contains the DMA register set occupying locations '40 to '77. These locations contain direct memory channels 0 to '37, respectively, that allow controllers to access memory with a minimum of processor intervention. Extended DMA, available only on the 6350, allows any even 32-bit location in the I/O segments to operate as a DMA channel pair.

Making a DMA Request

To perform a DMA transfer, a program must:

1. Set up a DMA cell
2. Tell the controller to perform the transfer

In regular DMA, a DMA cell is one 32-bit location in the register file, as shown in Figure 11-5. Through Extended DMA (for the 6350 only), any even 32-bit location in the I/O segments can also serve as a DMA cell. Bits 1 to 12 of a DMA cell location contain the two's complement of the total number of 16-bit quantities to be transferred. This means that the largest block of 16-bit quantities that can be transferred on a single channel is 4096; to transfer more requires more than one channel.

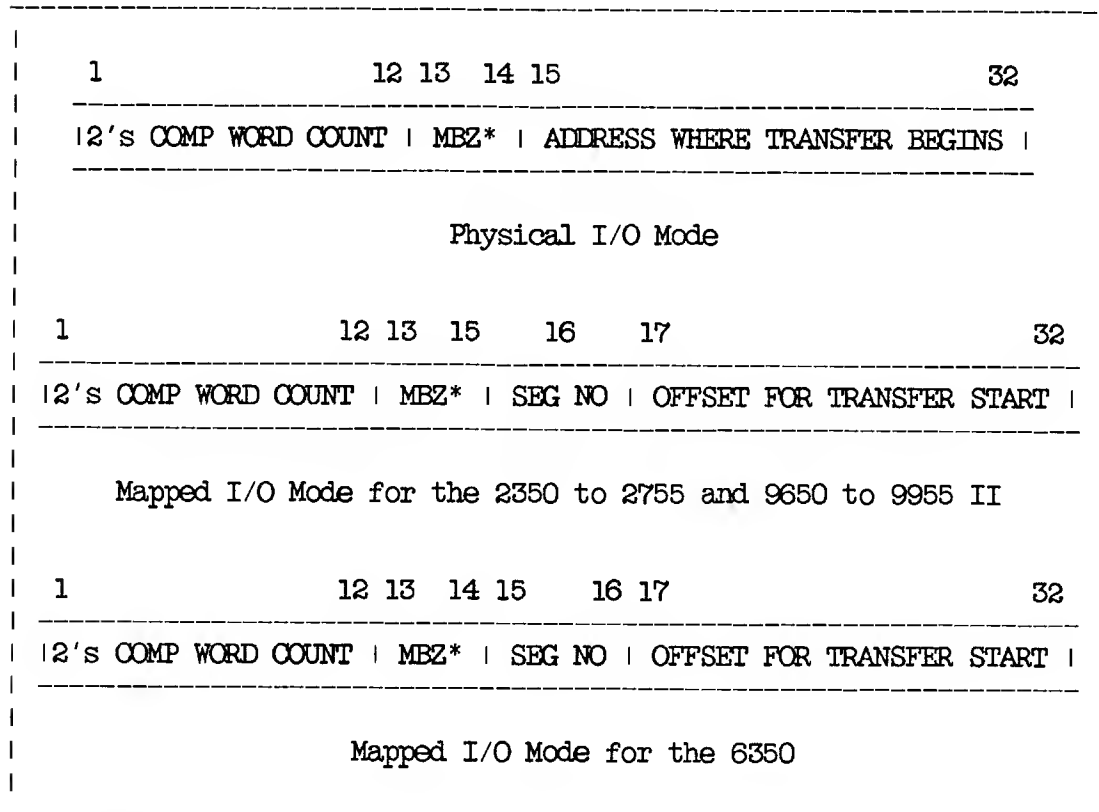
DMA on the 6350 can transfer 32-bit quantities as well as 16-bit ones, depending on the mode specified. The definition for bits 1 to 12 of a DMA cell location, however, remains unchanged. Thus, after the 6350 performs a single 32-bit transfer, it increments the count by 2. After a single 16-bit transfer occurs, all processors increment the count by 1.

The use of bits 13 to 32 depends on the machine and on whether mapped I/O mode or physical I/O mode is being used. In physical I/O mode, bits 13 and 14 are reserved but must be zero; bits 15 to 32 supply the physical address of the first location to transfer.

When mapped I/O mode is being used on the 6350, bits 15 and 16 select I/O segments 0 to 3; bits 17 to 32 specify the offset within the segment at which the transfer is to begin. Bits 13 and 14 are reserved but must be zero.

For mapped I/O on the 2350 to 2755 and 9650 to 9955 II, bit 16 selects I/O segment 0 or 1. Bits 17 to 32 specify the offset in the segment, and bits 13 to 15 are reserved but must be zero.

For the format of the DMA control word in mapped I/O mode for the earlier systems listed on page 1-1, see Appendix B.



* Must be zero.

Format of DMA Control Word
Figure 11-5

Servicing a DMA Request

When a controller wants to make a DMA transfer, it asserts a *DMx* request line to the processor. It specifies the type of transfer on the mode lines and the channel address on the BPA address lines. Normally, the processor acts on the request one microstep after the request arrives. If only one request is pending, the processor services it immediately. If more than one is pending, the processor services the request from the controller mounted in the lowest numbered I/O slot first, then it services the request from the controller in the next lowest slot, and so on.

When the processor pauses to service a request, it services all pending requests before resuming instruction execution or servicing an interrupt.

Once the processor has selected a request for service, it fetches the quantity to transfer and either sends them over the bus, or stores them at the address specified by the channel control words. For single 16-bit transfers, the processor then increments the values of the 16-bit quantity count and transfer address by 1; for single 32-bit transfers (6350 only), the count is incremented by 2. If the incremented value of this count is 0, the processor issues an EOR (end of range) signal. A count of any other value means that there is more data to transfer.

At the end of each request, the count specifies the number of 16-bit quantities left to transfer and the transfer address specifies the address of the next quantity to transfer. At the normal end of the transfer, the count contains a 0 and the transfer address specifies either: the address of the last 16 bits transferred plus 1, or the address of the last 32 bits transferred plus 2.

Burst Mode DMA

Burst mode DMA has two forms: 16-bit burst mode and 32-bit burst mode. Only the 6350 supports 32-bit burst mode DMA. Of the earlier systems listed on page 1-1, 16-bit burst mode is used only with the 750 and 850.

Burst mode DMA operations are similar to DMA transfers because they are both set up the same way. Like regular DMA, burst mode DMA sets up a DMA cell and tells the controller what to transfer. The difference is that burst mode DMA sends four quantities of data in each transfer, rather than just one. These may be four 16-bit quantities, or, for the 6350 only, four 32-bit quantities. This makes burst mode DMA efficient for transferring large blocks of data. After each transfer, the DMA range count and address are both incremented by either 4 (16-bit burst mode DMA) or 8 (32-bit burst mode DMA).

The data to be transferred can be arbitrarily aligned in memory. However, burst mode will operate at ordinary DMA rates unless the data is aligned as follows: for 16-bit burst mode, aligned on a 64-bit boundary with at least 64 bits left in the range; for 32-bit burst mode, aligned on a 128-bit boundary with at least 128 bits left in the range.

Controllers designed to make 16-bit burst DMA transfers can make 16-bit burst DMA requests via the mode lines at any time. On processors that do not implement 16-bit burst DMA, single 16-bit DMA transfers will be generated by the processor, instead of 16-bit burst DMA transfers.

On processors that do implement 16-bit burst DMA mode, a controller's request for a 16-bit burst DMA transfer will result in a 16-bit burst

DMA transfer if the data is 64-bit aligned and if there are at least 64 bits left to transfer. If both of these conditions are not met, the processor will generate 16-bit DMA transfers. These 16-bit single transfers may occur at the beginning of a transfer until 64-bit alignment is achieved or at the end of a transfer because there are less than 64 bits remaining to be transferred, or both.

The controller typically is not aware of the length of the transfer, its alignment in memory, or the type of CPU in the system. Therefore, the controller must be able to accept either 16-bit single DMA transfers or 16-bit burst DMA transfers from the CPU anytime that it makes a 16-bit burst mode transfer request.

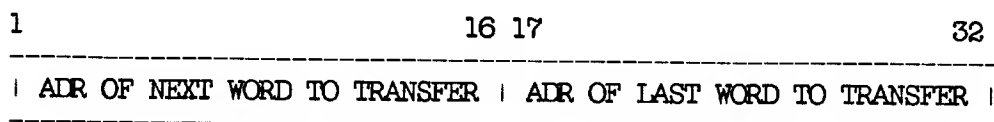
Extended DMA

Extended DMA operations, supported only by the 6350, are similar to ordinary DMA transfers with the following difference. In extended DMA, the DMA channel pair may be any even 32-bit location in memory in the supported I/O segments, rather than being restricted to the DMA register file. After setting up the channel pair, extended DMA operates in a manner identical to DMA. Burst and non-burst operations are permitted in this mode.

DMC

DMC operates in much the same way as DMA does. The differences are that DMC provides a total of 32,768 channels rather than just 32, and that data blocks can contain up to 128K bytes. Also, the DMC transfer rate is much slower than that for DMA since DMC performs three memory operations per transfer versus one for DMA.

DMC operations require a control word just as DMA operations do. The DMC control word, however, is not contained in the current register file, but in a 32-bit location in an I/O segment. Transfers must be in the same segment as the channel pair. Bits 1 to 16 of the control word contain the 16-bit address of the next 16 bits to be transferred; bits 17 to 32, the 16-bit address of the last 16 bits to be transferred. (See Figure 11-6.) The DMC control word must be aligned on an even 16-bit boundary.



Format of DMC Control Word
Figure 11-6

As in DMA service, a controller uses BPA to request the processor for memory access via a specified channel. When the processor can break its execution, it services any pending requests. If more than one request is pending, the processor services the request of the controller mounted in the lowest numbered slot first, then others in order of their priority.

Once it has selected a request to service, the processor either reads or writes the contents of the location specified in bits 1 to 16 of that channel's control word. After the read or write, the processor increments the contents of bits 1 to 16 by 1. If the value before the increment equals the contents of bits 17 to 32 in the control word, the processor issues an EOR signal. If the two values are not equal, then there is more data to transfer.

At the end of each request, bits 1 to 16 of the control word point to the next 16 bits to transfer. At the normal end of the transfer, bits 1 to 16 point to the last transferred location plus 1.

DMT

DMT transfers are used by controllers that do not need an external control word stored in memory or in the register file. Since the controller specifies all the information necessary to perform the transfer, all channel control functions can overlap with processor and memory functions at a speed equivalent to that of DMA. DMT transfers are useful when manipulating tumble tables and channel programs.

When a controller wants to request a DMT transfer, it uses BPA to ask the processor for memory access. When the processor can service the request, it transfers data to or from the controller. The address specified by the controller is either the source or the destination of the data to transfer, depending on the transfer direction.

Burst Mode DMT

Supported by the 6350 only, 16-bit burst mode DMT operations are similar to single DMT transfers because the controller specifies all information for the transfer, enabling channel control functions to overlap with processor and memory functions. The difference is that burst mode transmits four 16-bit quantities of data in each transfer, rather than just one. This makes burst mode efficient for transferring large blocks of data.

The controllers do not request burst mode transfer unless they have 64 bits or more of data to transfer. If the controllers have been doing a burst mode transfer but have, for example, 32 bits left, they must request ordinary DMT transfers. Controllers may only use this mode when they have been told to use it by the Operating System. This is a difference between burst DMT and burst DMA modes.

DMQ

Chapter 6, DATATYPES, defined queues, their parameters, and how they are manipulated. As noted there, one of their uses is as a storage device. DMQ operations use physical memory queues to hold data traveling between device and processor.

To make a DMQ request, the controller uses BPA to ask the processor for queue access via a selected QCB. (The QCB address specified over BPA must be aligned on a 64-bit boundary.) For an input operation, the processor adds the 16-bit contents of the specified address to the bottom of the queue (equivalent to an ABQ), if there is room. If there is no room, the processor sends an EOR signal to the controller.

For an output operation, the processor removes the first 16 bits from the top of the queue (equivalent to an RTQ) and transfers it to the specified address. If the queue contains no data, the processor issues an EOR to the controller, as well as 16 bits of zeroes. If the processor removes the last 16 bits from a queue, however, it does not signal the controller.

DMQ is fully interlocked with the queue manipulation instructions shown in Chapter 6.

DMX ADDRESS FORMATION

The process that results in the formation of the addresses of the data during DMx transfers begins with a controller driving the I/O bus address lines. When the addresses are received by the CPU, they are interpreted and used differently based on whether the CPU is in mapped or unmapped I/O mode and which DMx mode is being requested by the controller. The controller, in general, has no knowledge of what modes the CPU is operating in or of the CPU type.

Typically, there are two address formation processes that occur. The first forms the address of the channel control words. The second uses the channel control words to form the address of where the data is to be read from or written to. Table 11-10 shows how these addresses are formed and used in the the various DMx modes.

Table 11-10
DMX Address Formation

DMx Mode	Mapped or Not Mapped	Channel Control Word (CW) Address	Current Data Address
DMT	Mapped	None	Virtual: Segment # = BPA 00, 99 Offset = BPA 1 to 16 Physical: Virtual mapped through a TLB
DMT	Not Mapped	None	Physical: BPA (18 bits)
DMA	Mapped	If BPA < 32, in reg addressed by BPA + '40. If BPA >= 32, Virtual: BPA (18) Physical: mapped virt. through TLB	Virtual: Seg # = CW bits 15, 16 Offset = CW bits 17 to 32 Physical: Virtual mapped through a TLB Virtual: Seg # = CW bits 15, 16 Offset = CW bits 17 to 32 Physical: Virtual mapped through a TLB
	Not Mapped	If BPA < 32, in reg addressed by BPA + '40 If BPA >= 32, Physical: BPA (18 bits)	Physical: CW bits 15 to 32 Physical: CW bits 15 to 32
DMC	Mapped	Virtual: BPA (18 bits); Physical: mapped virt through TLB	Virtual: Seg # = Seg # of CW Offset = CW bits 1 to 16 Physical: Virtual mapped through a TLB
DMC	Not Mapped	Physical: BPA (18 bits)	Physical: CW bits 1 to 16
DMQ	Mapped	If QCB.V=0: QCB virtual is BPA (18 bits); QCB physical is virtual mapped through TLB	Physical: QCB bits 37 to 48 Top or Bottom Pointer
DMQ	Not Mapped	If QCB.V=0, phys. is BPA (18 bits)	Physical: QCB bits 37 to 48 Top or Bottom Pointer
DMQ	Mapped or Not	If QCB.V=1, invalid operation	

Notes to Table 11-10

For DMA: In not mapped I/O, if $BPA < 32$, the control word is in a register. In mapped I/O, if $BPA \geq 32$, the control word is in main memory.

For DMC: The control word is in main memory.

APPENDICES

A

Power-up

POWER-UP AND SYSTEM INITIALIZATION

All 50 Series processors perform the following steps in the sequence shown for power-up and system initialization.

1. Power becomes valid.
2. VCP (Virtual Control Panel) or maintenance processor conducts self tests.
3. CPU micro-diagnostics perform processor validation.
4. CPU initializes to the state shown in Table A-1.

Note

The failure of step 2, 3, or 4 stops the entire process and causes an error message to be displayed.

Table A-1
CPU Initialization Values

Element Initialized	Initialized Value
CRS (current register set)	0 (specifies RF2, the first user register set)
Registers in CRS	0, generally
All DMA (direct memory access) I/O registers but 6	Undefined
DMA register 6	0 (or 3) // 1000 (manufacturing test equipment)
Keys	0 (addressing mode now 16S)
Modals	0
Program counter	Ring 0, segment 0, offset '1000
RSVPTR (register save pointer)	0

B

Earlier Processors

The earlier processors are the following.

2250	850	750
650	550-II	550
500	450	I450
400	350	250-II
250	150	

This appendix discusses the implementation of the 50 Series architecture on these earlier systems, so called because they were produced earlier than the 9950.

The discussion of architectural topics in this appendix is presented in the order that they were presented in Chapters 1 through 11 of this guide. For example, the first section in this appendix is System Overview, which is the title of Chapter 1; the last section is Input-Output, the title of Chapter 11.

SYSTEM OVERVIEW

Chapter 1 presented an overview of the major units of the architecture common to all 50 Series processors. All 50 Series processors but the 850 have a single-stream architecture; the 850 has a dual-stream one. Both single-stream and dual-stream architectures as implemented on the earlier processors are discussed below.

Single-stream Architecture

A 50 Series processor with single-stream architecture can be divided into four major units as shown in Figure 1-1. These units are the cache and STLB, the control store, the execution unit, and the instruction unit. The following paragraphs describe each of these units as implemented in the earlier single-stream processors.

Cache and STLB: For the earlier processors, the cache and the STLB vary only in the amount of information that they can contain. For the 750 and 850, each cache entry contains information about four bytes of recently accessed physical memory, as do those of the 2350 to 9955 II. For the other earlier processors, each cache entry has information about two bytes. The STLB of all earlier processors contains the results of the last 64 virtual-to-physical address translations.

The Control Store Unit: The earlier processors support up to 64 Kbytes of ROM control store address space.

The Execution Unit: The execution unit elements appear in Figure 1-2. These elements are: an integer arithmetic logic unit (ALU), a decimal ALU, a floating-point unit, and register files. Of these elements, only the number of register files varies according to processor type. The earlier processors have four register files: one microcode and system status register file, two user register files, and one direct memory access file.

The Instruction Unit: Of the earlier processors, only the 750 and 850 have an instruction unit, designed to speed up execution by processing information about instructions before execution. When the execution unit is performing an add or similar operation for instruction n , the instruction unit is working on the next two instructions. This unit is decoding instruction $n+1$, calculating its address, and determining what registers, if any, are to be accessed. It is also fetching instruction $n+2$ from the cache so that it can be decoded when instruction $n+1$ begins to execute. This means that, in most cases, when the execution unit finishes one operation, the instruction unit has already done the calculations necessary to allow the execution unit to perform the next instruction without delay.

Dual-stream Architecture

The 850 processor has a dual-stream version of the 50 Series architecture. This dual-stream nature enables the 850 to provide 60% to 80% more service than the 750. Figure B-1 shows a block diagram of the 850 dual-stream architecture.

Instruction Stream Units: The 850 contains two Instruction Stream Units (ISUs), each of which is similar in capabilities to a 750 CPU. Each ISU executes an independent stream of instructions simultaneously, synchronized by a Stream Synchronization Unit (SSU). (See below.) Each ISU is responsible for:

- Full instruction decode
- Effective address calculation
- Instruction execution
- Calculating data for the anticipated next instruction

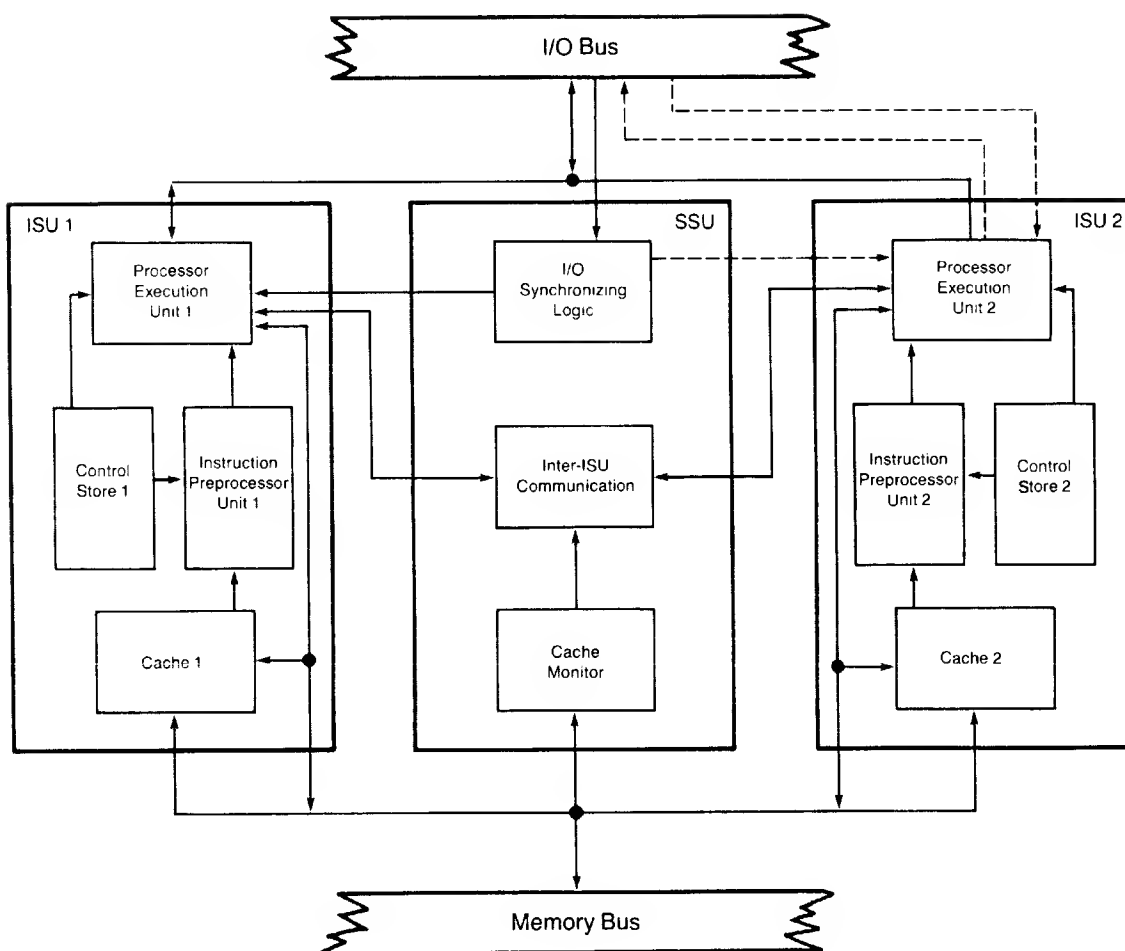
The four blocks shown in each ISU contain the same elements and perform the same functions as those described in the first part of this chapter.

The two ISUs share one copy of the operating system. PRIMOS is reentrant and can run on either ISU (as can any user program), so duplicate copies are not needed. System actions are also simplified, since there is no need to check for or handle discrepancies caused by different versions of the operating system.

Stream Synchronization Unit: The primary task of the SSU is to prevent improper information from being loaded into the cache of either ISU. It does this by maintaining a list of the contents of both caches. When data is written into either cache, the SSU detects it and invalidates the contents of the appropriate entry in its list of cache contents. This means that the SSU always knows which cache locations contain current information and which do not.

When a cache location in one of the ISUs contains information that is out of date, the SSU notifies that ISU of the discrepancy. That ISU invalidates the stale entry, thus forcing a memory read to the current information the next time that location is referenced.

In addition to synchronizing cache references, the SSU also coordinates references to memory and system handlers. The two ISUs share one main memory, one operating system, and one copy of several system handlers. To ensure that these resources are used effectively and efficiently, the SSU contains four locks.



Dual-stream Architecture
Figure B-1

The process exchange lock aids the process exchange mechanism (see Appendix C) to transfer control smoothly between processes on both ISUs. The queue lock controls situations in which simultaneously executing queue instructions (one on each ISU) are vying for access to a single queue. It ensures that both instructions get access, but that neither one interrupts or interferes with the other. The check lock allows only one ISU to signal a check at a time, thus guaranteeing that the single set of check handlers services all checks. The fourth lock, the mutual exclusion lock, can be used by software to prevent both ISUs from trying to access a particular procedure or piece of data at the same time.

Diagnostic operations and communications between ISUs are also handled through the SSU. The former feature aids in system monitoring and testing; the latter enhances the 850's ability to execute independent instruction streams without high system overhead.

PHYSICAL AND VIRTUAL MEMORY

Chapter 2 discussed the characteristics of the 50 Series virtual and physical memory. Virtual memory is the same for all 50 Series processors. There are three types of physical memory on the 50 Series: cache, main memory, and disk. As implemented in the earlier processors, the cache size and hit rate vary according to processor type as shown in Table B-1.

Table B-1
Cache Sizes and Hit Rates for Earlier Processors

System	Cache Size	Hit Rate
150 to 500	2 Kbytes	85%
550-II to 650	8 Kbytes	90%
750	16 Kbytes	95%
850	32 Kbytes	95%
2250	2 Kbytes	85%

The Memory Management section in this appendix discusses the cache entry format implemented on the earlier processors. The main memory of the earlier processors is comprised of one-megabyte E Series memory boards. Earlier processors such as the 2250 that have a 16-bit wide data path require a minimum of one memory board. The earlier processors such as the 750 that have a 32-bit wide data path require a minimum of two memory boards. The total memory capacity of the earlier processors, like that of the 2350 to 2655, 9650, and 9655, is 8 megabytes. Disk memory is the same for all 50 Series processors.

ADDRESSING

Chapter 3 discussed the kinds and modes of addressing supported by the 50 Series processors. Virtual address components and instruction formats are the same for all 50 Series processors. The earlier processors support four of the five types of address formation: direct, indexed, indirect, and indirect indexed. Neither general register relative (GRR) address formation nor C language pointers are supported on the earlier processors.

All addressing modes are supported by all 50 Series processors. For the earlier processors, the addressing range of 32I mode long is four segments because GRR is not supported.

In the tables of Chapter 3, only two vary according to processor type: 64V mode address formation for nonindexing instructions, and address trap action for short 64V mode instructions.

Table B-2 shows 64V mode address formation performed by the earlier processors for the nonindexing instructions DFLX, FLX, JSX, LDX, LDY, QFLX, STX, and STY.

Table B-2
64V Mode Address Formation for Nonindexing Instructions

I	X	Y	750 and 850	Rest of Earlier Processors
0	0	0	Direct	*Direct
0	0	1	Direct	Index by Y
0	1	0	Direct	*Direct
0	1	1	I(A)	I(A+X)
1	0	0	I(A)	I(A+Y)
1	0	1	I(A)	*I(A)
1	1	0	I(A)	I(A+X)
1	1	1	I(A)	*I(A)

Notes to Table B-2

- * These modes should be used to ensure consistent behavior across processors.

The symbol A in Table B-2 represents the value calculated from the base register (PB, SB, LB, or XB) and displacement in the instruction.

Table B-3 shows how the earlier processors take address trap action for short 64V Mode instructions.

Table B-3
Address Trap Action for Short Format
Instructions, 64V Mode

I	X	S	D	Action
0	0	0	'0 to '7	Takes address trap.
0	0	0	'10 to '37	Takes address trap only if segmentation is off.
0	0	0	'40 to '377	Cannot take address trap.
0	0	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.
0	1	0	'0 through ATR	Takes address trap if D+X is within the ATR. If D+X is outside the ATR, the EA is SB(seg #) D+X (750 and 850*), or SB(seg #) D+X+SB(word #) (rest of earlier processors).
			From ATR to '377	Cannot take address trap; EA is SB+D+X (750 and 850*). Rest of earlier processors take address trap if D+X is within the ATR.
			'400 to '777	Cannot take address trap.
0	1	1	-'340 to +'377	Takes address trap if EA (P+D+X) is within the ATR.
1	0	0	'0 to '777	Takes address trap if D is within the ATR.**
1	0	1	-'340 to +'377	Takes address trap if EA ((P+D)) is within the ATR.**
1	1	0	'0 to '777	Takes address trap if D<'100 and D+X is within the ATR.**
1	1	1	-'340 to +'377	Takes address trap if EA (P+D) is within the ATR.**

Notes to Table B-3

* Same action taken for the 2350 to 9955 II.

** The indirect address also takes an address trap if EA is within the ATR.

MEMORY MANAGEMENT

Chapter 4 told how a virtual address is translated into a physical address. This included a discussion of memory management data structures, accessing the STLB and cache, and address translation. The implementation of these items varies for the earlier processors as described below.

Memory Management Data Structures

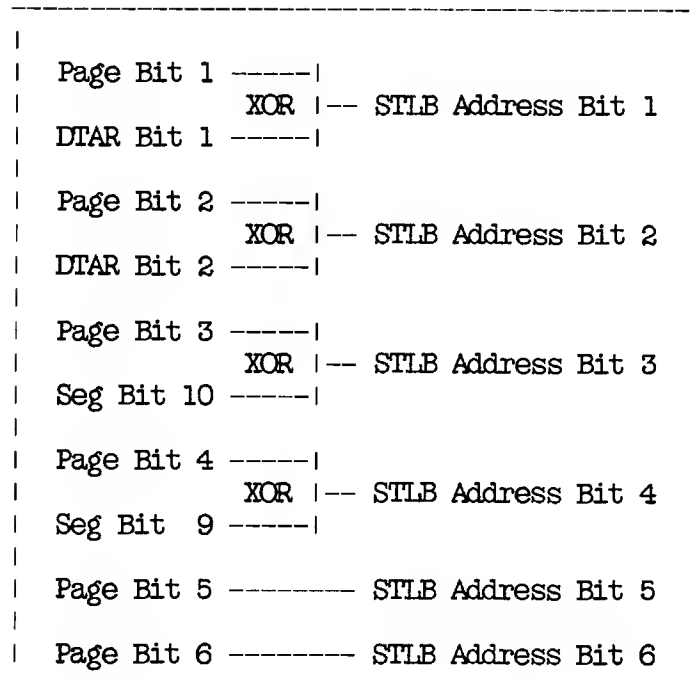
As discussed in Chapter 4, all 50 Series processors have the following memory management data structures: an STLB, a cache, four DTARs, many SDTs, and either HMAPs or PMTs. The earlier processors have HMAPs, as do the 2350 to 2655, 9650, and 9655; all other 50 Series processors have PMTs.

The STLB of the earlier processors has 64 entries whose format is shown in Figure B-2. The meaning of the STLB entry contents is identical to that contained in Table 4-3.

1	2	3	4		6	7		9	10		21	22	33	34		45
<hr/>																
V M S	RING 1				RING 3				PROC ID			SEG			PHYS ADR	

STLB Entry Format for the Earlier Processors
Figure B-2

The bits used by the earlier processor in their hashing algorithm are DTAR bits 1 and 2, segment bits 9 and 10, and page bits 1 to 6. The hashing algorithm of the earlier processors is shown in Figure B-3.



STLB Hashing Algorithm for the Earlier Processors
Figure B-3

The cache entry format of the earlier processors varies according to processor type. The cache entry format of the 750 and 850 is the same as that of the 2350 to 2655, 9650, and 9655 because it includes 32 bits of data. For the rest of the earlier processors, the cache entry format includes 16 bits of data. Both of these formats are shown in Figure B-4.

1	1	12	1	32
V	PHYSICAL PAGE NUMBER		DATA	

750 and 850 Cache Entry Format
(Like That of 2350 to 2655, 9650, and 9655)

1	1	12	1	16
V	PHYSICAL PAGE NUMBER		DATA	

Cache Entry Format of Other Earlier Processors

Number of Bits	Mnemonic	Description
1	Valid	The cache holds valid data when this bit contains 1.
12	Physical Page Number	Specifies the number of the physical page that contains the specified location.
16 or 32	Data	Contains a copy of the contents of a location in physical memory.

Cache Entry Format of the Earlier Processors
Figure B-4

The DTARs and SDTs are the same for all 50 Series processors.

The HMAPS used by the earlier processors are the same as those used by the 2350 to 2655, 9650, and 9655. The HMAP format is shown in Figure 4-12.

Accessing the Cache and STLB

As discussed in Chapter 4, a slightly different set of actions is performed to access the cache and STLB depending on whether the operation is a read or a write. Write memory access is the same for all 50 Series processors. Read memory access varies only in the number of virtual address bits used to reference an entry in the cache index.

The 750 and 850 use bits 20 to 32 of the virtual address, as do the 2350 to 2655 and 6350 to 9950. These bits are the least significant three bits of the page field and the 10-bit offset field.

The I450 and 550-II hardware uses virtual address bits 21 to 32. These bits are the least significant two bits of the page field and the 10-bit offset field.

For the other earlier processors, the hardware uses virtual address bits 23 to 32 as the address of a cache index entry. These bits are the 10-bit offset field.

The few page field bits mentioned above create a virtually mapped cache described in Chapter 11 and the Input-Output section of this Appendix.

Address Translation

Address translation for the earlier processors is the same as for the 2350 to 2655, 9650, and 9655. This process is diagrammed in Figure 4-16.

CONTROL INFORMATION AND RESTRICTED INSTRUCTIONS

Chapter 5 discussed the modals, keys, and restricted instructions. The modals and restricted instructions are the same for all 50 Series processors. The keys are also the same for all 50 Series systems, with the exception of bits 12 to 14 of the V mode and I mode keys.

For the format of the V and I mode keys, see Figure 5-4. Bits 12 and 13 (ASCII-8 and RND, respectively) are disregarded on the earlier processors. Bit 14, the P850 bit, is used only by the 850 processor. (For further details of the P850 bit, see Appendix C which describes process exchange on the 850.)

DATATYPES

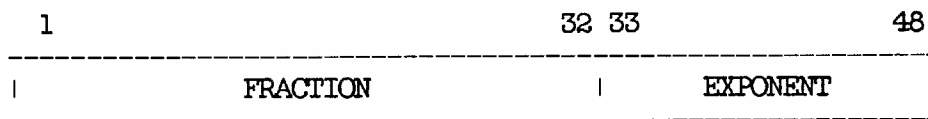
Chapter 6 discussed the datatypes supported by the 50 Series processors: fixed point data, floating-point numbers, decimal integers, character strings, and queues. With the exception of floating-point numbers, each of these datatypes is represented in the same way for all 50 Series processors and has the same operations and instructions available to manipulate each type.

Some aspects of floating-point representations and operations vary according to processor type. The floating-point features for the earlier processors are presented below.

The earlier processors support single precision (SP) and double precision (DP) operations, but not quad precision (QP).

The memory format of SP and DP floating-point numbers is the same for all 50 series processors as is the double precision accumulator (DAC); these are shown in Figure 6-4.

The single precision accumulator (FAC) format for the earlier processors 750 and 850 is the same as shown in Figure 6-4. For the rest of the earlier processors, however, the FAC format is as shown below in Figure B-5.



SP Floating-point Accumulator Format for Earlier Processors
Figure B-5

Floating-point overflow and underflow are the same for all 50 Series processors.

The use of guard bits during normalization varies according to processor type. Floating-point multiply instructions for the 550-II, 650, 750, and 850 keep guard bits for normalization use; for all other floating-point instructions, however, these processors shift in only zeroes during normalization of the results and do not use guard bits. For the rest of the earlier processors, no guard bits are saved; this processor shifts in only zeroes during normalization.

Floating-point rounding varies according to processor type. See Table B-4 below for rounding on the earlier processors.

Table B-4
Rounding Prerequisites and Procedures of the Earlier Processors

Type	550-II, 650, 750, and 850	Rest of Earlier Processors
SP	Add, subtract, multiply: FRN compiler option rounds result just before store. (See Store below.)	Add, subtract, multiply: FRN compiler option rounds result just before store. (See Store below.)
	Divide: Always rounds. 33 fraction bits are generated for rounding to 32.	Divide: Rounding never done.
	Store: FRN rounds and normalizes just before the store. If FAC bit 25 = 1, then add 1 to bit 24 and zero rest of FAC fraction.	Store: FRN rounds and normalizes just before the store. If FAC bit 25 = 1, then add 1 to bit 24 and zero rest of FAC fraction.
	Compare and Skip: Rounding never done.	Compare and Skip: Rounding never done.
DP	Divide: 49 fraction bits generated for rounding to 48.	Divide: Rounding never done.
	Other instructions: Rounding never done.	Other instructions: Rounding never done.

The discussion of normalized versus unnormalized operands in Chapter 6 applies equally to all 50 Series processors. Divide produces indeterminate results on the 550-II, 650, 750, and 850 (as well as the 2350 to 9955 II) when confronted with unnormalized numbers.

Floating-point accuracy and precision vary according to processor type. Tables B-5 and B-6 summarize floating-point accuracy and precision for the earlier processors.

Table B-5
Floating-point Instruction Accuracy for the Earlier Processors

Instruction	750 and 850	550-II and 650	Rest of Earlier Systems
FAD	48	32	32
DFAD	48	48	48
FSB	48	32	32
DFSB	48	48	48
FMP	48+	32+	29
DFMP	48+	48+	45
FDV	31*	31*	30
DFDV	47*	47*	46

Notes to Table B-5

+ means 2 extra guard bits are used.

* means rounding is always performed.

The values in Table B-5 refer to the number of fraction bits guaranteed to be accurate for the indicated processor. This number includes the sign bit because the fraction represents a two's complement value. Other manuals may emulate a sign-magnitude representation in statements about accuracy. A sign-magnitude representation requires a 1 to be subtracted from all entries in this table. Worst case normalization is included in all results. The accuracy of an infinite precision result lies closer to the number indicated than to either of its neighboring representations.

Table B-6
Floating-point Precision for the Earlier Processors

Precision	750 and 850	550-II and 650	Rest of Earlier Systems
Fraction Bits:			
Memory	24/48/--	24/48/--	24/48/--
Accumulator	48/48/--	32/48/--	32/48/--
Exponent Bits:			
Memory	8/16/--	8/16/--	8/16/--
Accumulator	16/16/--	16/16/--	16/16/--
Guard Bits	2 for multiply	2 for multiply	None
Rounds Automatically	For divide	For divide	No

Notes to Table B-6

The number of fraction and exponent bits is shown in SP/DP/QP form.

The fraction values in Table B-6 refer to the number of fraction bits for the indicated processor. This number includes the sign bit because the fraction represents a two's complement value; other manuals may emulate a sign-magnitude representation. A sign-magnitude representation requires a 1 to be subtracted from all fraction entries in this table.

750 and 850 Systems: The 750 and 850 processors operate in DP even when executing SP instructions. Floating load instructions zero accumulator bits 25 to 48. SP add, subtract, and multiply instructions do not truncate accumulator fractions to 32 bits, resulting in an additional 16 bits of precision. The multiply instruction keeps extra bits of precision that are used during normalization.

In an SP divide instruction, one fraction is 48 bits and the other is 24 bits. This instruction generates 33 fraction bits and rounds to 32 before placing the result in the SP accumulator. A DP divide instruction, however, generates 49 fraction bits and rounds to 48.

550-II and 650 Systems: A 550-II or 650 system has a separate double-precision hardware floating-point unit. These systems insert zeroes in fraction bits 25 to 48 of an SP memory argument before

loading the accumulator. They also zero fraction bits 33 to 48 for arguments from the SP accumulator. All arithmetic operations are then performed in DP.

Fractions are truncated to 32 bits to place the results in the FAC, leaving the low order 16 bits alone in the overlapped DAC. Storing a number in SP memory truncates a number further to 24 bits. A multiply instruction alone preserves two extra bits of precision for use in normalization.

A divide instruction automatically generates an extra fraction bit for rounding the result to 32 bits (SP) or 48 bits (DP).

A single precision floating load instruction always zeroes accumulator bits 25 to 48 before actually loading the number for systems with PRIMOS Rev. 18 or above.

Other Earlier Systems: When an SP number is loaded from memory to the accumulator, zeroes are placed in FAC fraction bits 25 to 32. After performing a floating-point operation, the FAC fraction contains a 32-bit result. To store this result in SP memory, the processor truncates bits 25 to 32 but leaves bits 33 to 48 alone.

DP memory and accumulator fractions both have a capacity of 48 bits, so no bits of precision disappear when transferring DP numbers from one place to the other.

A single precision floating load instruction always zeroes accumulator bits 25 to 48 before actually loading the number for systems with PRIMOS Rev. 18 or above.

Converting Datatypes

The conversion of floating-point numbers to integers and vice versa is the same for all 50 Series processors with the exception that the earlier processors do not have quad precision. Therefore, the earlier processors cannot convert quad precision numbers to integers and vice versa.

ALTERING SEQUENTIAL FLOW

Chapter 7 discussed the instructions that alter the sequential flow of a program: branch, skip, and jump. These instructions are the same for all 50 Series processors.

STACKS AND PROCEDURE CALLS

Chapter 8 discussed stacks and their management, entry control blocks, indirect pointers, gate access, making a procedure call, and the ARGV and PRTN instructions. All of these items are the same for all 50 Series processors.

PROCESS EXCHANGE

Chapter 9, PROCESS EXCHANGE, discussed the process exchange mechanism (PXM) on single-stream 50 Series processors. Since the 850 has a dual-stream architecture, process exchange on this earlier processor is discussed in Appendix C.

The main elements of the PXM are the same for all single-stream 50 Series processors. Thus, they have the same process control blocks, ready list, wait lists, WAIT instruction, NOTIFY instructions, and dispatcher.

As discussed in Chapter 9, three types of register files form the register set: microcode and system status register files; user register files; and a DMA channel register file. The number of register files in a register set, however, varies according to processor type.

There are four register files for the earlier processors: two user register files, one microcode and system status register file, and one DMA channel register file. The allocation for these register files appears in Table B-7.

Table B-7
Register File Allocation for the Earlier Processors

Register File	Absolute Locs	Use
RFO	0 to '37	Microcode scratch and system registers
RF1	'40 to '77	32 DMA channels
RF2	'100 to '137	User register set 2
RF3	'140 to '177	User register set 3

Note to Table B-7

The two user register sets listed in this table are called user register sets 2 and 3 to correspond with their register file numbers RF2 and RF3.

The format of each user register file is the same for all 50 series processors and is shown in Table 9-5. The DMA channel register file, shown in Table 9-6, is also the same for all 50 Series processors. The format of the single microcode scratch and system status register file for the earlier processors appears in Table B-8.

Table B-8
Microcode Register File, RFO, for the Earlier Processors

Loc	Contents	Loc	Contents
0	TR0	'20	ZERO, ONE
1	TR1	'21	PBSAVE
2	TR2	'22	RDMX3
3	TR3	'23	RDMX4
4	TR4	'24	C377
5	TR5	'25	MINUS1, MINUS2
6	TR6	'26	WWADIR
7	TR7	'27	DSWPARTY
'10	RDMX1	'30	PSWPB
'11	RDMX2	'31	PSWKEYS
'12	USCADDR*,REOIV#	'32	PPA, PCBA
'13	RSGT1	'33	PPB, PCBB
'14	RSGT2	'34	DSWRMA
'15	REOC1	'35	DSWSTAT
'16	REOC2	'36	DSWPB
'17	---, RATMPL#	'37	RSAPVTR

* Used only for the 750 and 850 systems.

The locations for REOIV and RATMPL are switched on the 2250, 250, 400, and 550-II.

The nature of the process interval timer varies according to processor type. The 550-II, I450, and 850 use a timer accurate to the microsecond, as do the 2350 to 9955 II. These processors also support the two instructions that manipulate the process timer as shown in Table 9-12.

The process interval timer of the other earlier processors is accurate to the millisecond.

As discussed in Chapter 9, the basic steps of the dispatcher operation are the same for all 50 Series processors. These steps are as follows.

1. Turn off the process interval timer.
2. Choose a process to run.
3. Select a user register set for that process.
4. Turn the process interval timer back on.

Step 1 varies only in the number of bits used for the process interval timer in location '30 of the current register set. Bits 1 to 26 are used for the I450, 550-II, and 850, as is the case for the 2350 through 9955 II. Bits 1 to 16 are used for the other earlier processors.

Step 2 is the same for all 50 Series processors.

The operation of Step 3 for the earlier processors is shown in Figure B-6. For the earlier processors, the dispatcher makes the other user register set the current register set.

Step 4 is the same for all 50 Series processors.

INTERRUPTS, CHECKS, FAULTS, AND TRAPS

Chapter 10 discussed interrupts, faults, checks, and traps. The following paragraphs discuss the implementation of these breaks in execution for the earlier processors.

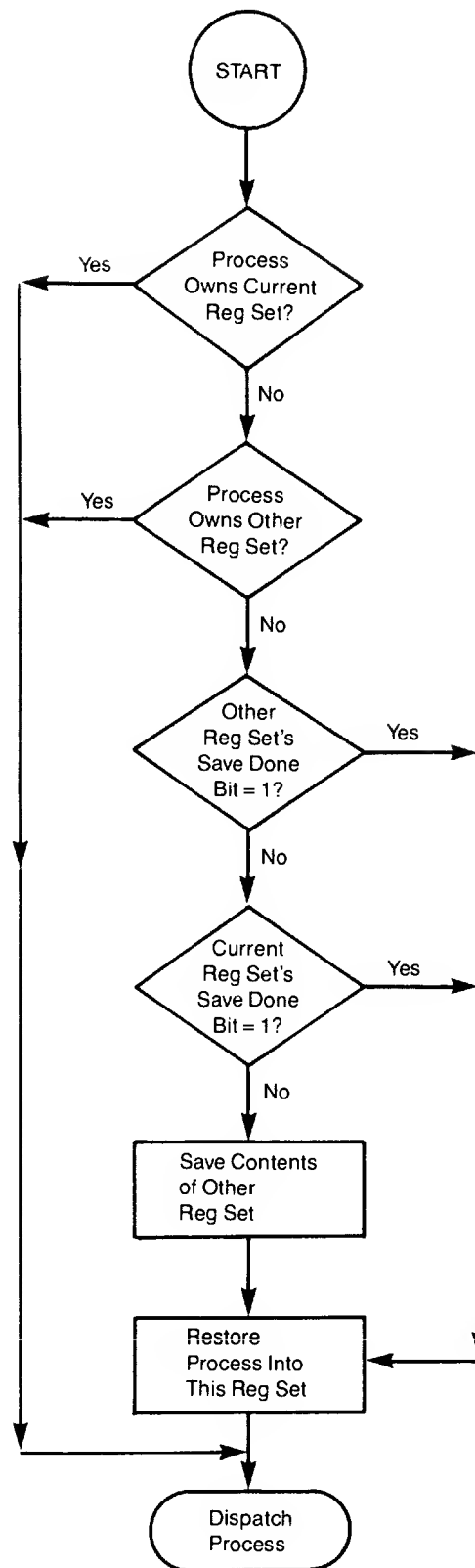
Interrupts

The two types of interrupts are external and memory increment. External interrupts are the same for all 50 Series processors. Memory increment interrupts are supported only on the earlier processors, and are described below.

Memory Increment Interrupt: Service for this interrupt is always the same, regardless of the process exchange mode. The processor uses the address supplied by the controller as a 17-bit offset into either of the I/O segments, 0 or 1 (if in mapped I/O). This offset addresses a halfword whose contents the processor increments by 1. If the incremented value does not equal 0, the processor does nothing more and returns.

If the incremented value does equal 0, the processor generates an end-of-range (EOR) signal on the I/O bus and returns. The requesting device typically generates an external interrupt when the EOR is generated.

Unlike the external interrupt, the memory increment interrupt cannot be masked out and can occur at any fetch cycle break.



Register Set Allocation Algorithm for the Earlier Processors
Figure B-6

Returning From a Memory Increment Interrupt: While the PXM mode does not affect service of this interrupt, it does determine where the processor returns to upon service completion. When the PXM is enabled, the processor returns to the fetch cycle or the dispatcher, depending on where the interrupt occurred. In the case of the dispatcher, the processor always returns to the top of the dispatcher and does not change the PB or KEYS.

When the PXM is disabled, the processor always returns to the fetch cycle.

Standard Interrupt Mode: Standard mode interrupts, although supported by all CPUs, are not used by PRIMOS. Current processors and controllers support this interrupt mode, but future ones will not. Obviously, the use of standard interrupt mode is strongly discouraged.

During standard mode I/O Interrupts, the I/O bus address lines received by the CPU are interpreted and used differently based on the mode that the CPU is in. The controller has no knowledge of which of these modes the CPU is operating in or of the CPU type. The controller actions during interrupt cycles are always the same.

When performing interrupts the CPU reaches the interrupt response code by performing an indirect JMP or JST through offset '63 as shown in Table B-9.

Table B-9
Address Formation for Standard Interrupt Mode

Interrupt Mode	PXM Enabled?	Mapped I/O?	Indirect Jump Address	Instruction Emulated
Standard	No	No	Physical = '63	JST '63,*
#Standard	Yes	No	Physical = '63	JMP '63,*
#Standard	No	Yes	Segment 0, offset = '63	JST '63,*
#Standard	Yes	Yes	Segment 4, offset = '63	JMP '63,*

means not recommended.

In standard interrupt mode, only one IRC can execute at a time, so the IRC has nothing to clear or save (other than the contents of any registers it intends to use) before reenabling interrupts. As in vectored mode, the IRC completes the rest of its operation and transfers control back to the location whose address is contained in the first IRC location.

Faults

Of the twelve types of faults, eleven are supported by all 50 Series systems: RXM, process, page, SVC, UII, ILL, access, arithmetic, stack, segment, and pointer. Only semaphore faults are not supported by the earlier processors.

Fault handling is the same for all 50 Series processors, excluding the handling of integer exceptions. For a 750 or 850, from one to four instructions are executed before the integer overflow exception occurs and the fault is taken. (The only exception to this is in the case of divide by zero, which always points to the next instruction.)

For the other earlier processors, when an integer overflow exception occurs, the resulting fault is taken before the next instruction is started. The program counter points to the next instruction suitable for execution. If, however, an ECCC check becomes pending at the same time as the integer exception, that integer exception will be lost.

Table 10-10 shows arithmetic exception codes. In this table only the FCODEH contents for a decimal overflow exception vary according to processor type. These contents are \$700 for the 750 and 850 as well as the 2350 to 9955 II. For the other earlier processors, the FCODEH content on a decimal overflow is \$704.

Checks

Four of the six types of checks apply to all 50 Series processors: power failure, memory parity error, machine check, and missing memory module. The earlier processors do not have environmental checks or recoverable machine checks.

The check handler is the same for all 50 Series processors in that it uses a check header, check vectors, diagnostic status words, and the MCM field of the modals. Of these items, only the diagnostic status words vary according to processor type. The DSWPARITY of the 750 and 850 appears in Table B-10. No other earlier processor has a DSWPARITY. Table B-11 shows the DSWSTAT for the earlier processors. Tables 10-20 and 10-21 show the DSWRMA and DSWPB, respectively, for all 50 Series processors.

Table B-10
Format of DSWPARITY Register for the 750 and 850

Bits	Name	Description
1	RPA Parity Error, Type 1	If 1, the control store has detected a parity error as follows: DMx input E6: BPD or Burst- R0, R2 DMx input E5: BPD or Burst- R0, R1, R2, R3 DMx output: BMD.
2	RPA Parity Error, Type 2	If 1, DMx input E6: BPD or Burst- R1, R3 DMx input E5: BPD DMx output: BMA.
3	Burst-mode DMx Parity Error	If 1, the control store detected a DMx burst mode parity error.
4	DMx I/O Parity Error	Setting specifies that the control store detected a DMx parity error as follows: 0: DMx input 1: DMx output.
5 to 7	J Board Parity Errors	The J board detected a parity error as follows: 000: peripheral reports BPD error (output) 001: base register file high 010: memory reports BMD error (write) 011: prefetch buffer address 100: peripheral reports BPA error (output) 101: base register file low 110: memory reports BMA error 111: prefetch buffer instruction.
8	RCM Parity Error	If 1 and no board reported an error, then an RCM parity error has been detected.
9	ECCC Error	If 1, memory detected an EOC uncorrectable error on read.
10	Prefetch Board Parity Error	If 1, prefetch board parity error.
11	BPA Input Parity Error	If 1, BPA input parity error (DMx or interrupt).
12	RDX Parity Error	If 1, RDX parity error when most recently closed.
13	Register File Parity Error	If 1, register file parity error.
14	REA Parity Error	If 1, REAH or REAL parity error.

Table B-10 (continued)
Format of DSWPARITY Register for 750 and 850

Bits	Name	Description
15	DMx Cycle Parity Error	If 1, parity error occurred during DMx cycle.
16	AP Board Parity Error	If 1, AP board detected parity error.
17	C Board Parity Error	If 1, C board detected parity error.
18	BMD Input Even Parity Error	If 1, BMD input even word parity error.
19	BMD Input Odd Parity Error	If 1, BMD input odd word parity error.
20	Missing Memory Module	If 1, missing memory module at cache miss.
21	BMA Parity Error	If 1, memory detected BMA parity error at cache miss.
22	RMA Increment	If 1, RMA was incremented at time of parity error (cache miss).
23	BMA15 Indicator	Setting of BMA15 indicator at time of parity error (cache miss).
24	BMA16 Indicator	Setting of BMA16 indicator at time of parity error (cache miss).
25	ECCU Error	If 1, memory reports an ECC uncorrectable error on a cache miss.
26	ECCC Error	If 1, memory reports an ECC correctable error on a cache miss.
27	Cache Index Parity Error	If 1, cache index parity error on cache read.
28	Cache Data Odd Word Parity Error	If 1, cache data odd word parity error on cache read.
29	Cache Data Even Word Parity Error	If 1, cache data even word parity error on cache read.
30	Cache Cycle Purpose	Specifies the purpose of the cache cycle at the time of the error: 0: prefetch 1: execute
31 to 32	---	Currently unused.

Table B-11
Format of DSWSTAT Register for All Earlier Processors

Bits	Name	Description
1	Check Immediate	If 1, the check was taken immediately.
2	Machine Check	If 1, a machine check occurred.
3	Memory Parity	If 1, a memory parity error caused the check.
4	Missing Memory Module	If 1, a missing memory module error caused the check.
5 to 7	Machine Check Code	The hardware detected the cause of the trap as follows: 000: peripheral reports BPD error (output) 001: base register file high 010: memory reports BMD error (write) 011: prefetch buffer address 100: peripheral reports BPA error (output) 101: base register file low 110: memory reports BMA error 111: prefetch buffer instruction
8	RCM	Control unit memory -- this bit is reset when an error is detected.
9	ECCU	If bits 3 and 9 are both 1, the memory parity error was ECC uncorrectable.
10	ECCC	If bits 3 and 10 are both 1, the memory parity error was ECC correctable.
11	BUNV	If 1, the RP backup count in bits 12 and 13 is not valid.
12 to 14	RBPUP	Specifies the RP backup count, which is the amount DSWPB was incremented in the current instruction.
15	DMx Operation	If 1, a DMx transfer was in progress when the error occurred.
16	I/O Operation	If 1, an I/O operation was in progress when the error occurred.
17 to 22	ECC Syndrome Bits	If a memory parity error occurred, these bits describe the error. See Table 10-29 of Chapter 10.
23	---	Currently unused.
24	Memory Module Number	If a memory error occurred, this bit identifies the interleaved memory module that contained the error (bit 15 of address at time of error).
25	RMA Invalid	If 1, the contents of DSWRMA are invalid.
26*	U-verify Pass	U-verify pass number as follows: 0: first pass (check mode off) 1: second pass (check mode on)
27 to 32*	U-verify Test Failure	If set, contains the number of a failed u-verify test.

Note to Table B-11

- * Valid for 750 and 850. For rest of 50 Series: bit 26 is unused, bit 27 is the u-verify pass number, and bits 28 to 32 are the u-verify test failure number.

Table B-12 lists those checks that cause a microcode trap and describes the actions that occur for the earlier processors.

Table B-12
Check-produced Traps and Their Actions for the Earlier Processors

Event	Actions
Missing Memory Module, EOC Uncorrectable, or Machine Check during I/O (DMx, PIO, interrupt processing, excepting machine check for RCM parity)	Sets end-of-instruction flag to 1; sets REOIV to the proper offset or vector; sets MCM to 00; executes microcode return to the trapped microcode step. Correctable memory errors are ignored during I/O.
EOC Correctable Error (not during I/O)	Sets end-of-instruction flag to 1; sets REOIV to the proper offset or vector; sets MCM to 2; executes microcode return to the trapped microcode step.
Power Failure	Action is deferred until the next fetch cycle, and then a check is taken.
All other checks	Software check occurs immediately.

Traps

Traps and their priorities are described in Table 10-26. The operation of these traps is the same for all 50 Series processors with the following exceptions for the earlier processors. Diagnostic processor interrupts are invalid for the 750 and 850. Memory increment interrupt traps are used only on the earlier processors. (This trap occurs at the point where a controller requested service.) Cache or STLB parity error traps as well as hard parity error traps are not used on the earlier processors.

The interpretation of syndrome bits used in identifying bit errors in the earlier processors is shown in Table 10-29.

For the earlier processors, the additional software break caused by traps is the same as shown in Table 10-30. In addition, upon a memory increment interrupt trap, the interrupt occurs. (Memory increment interrupts are supported only on the earlier processors.)

Interval Clock

The earlier processors use a 500 Hz interval clock that generates a timing pulse every 2 milliseconds.

INPUT-OUTPUT

Chapter 11, INPUT-OUTPUT, discussed Programmed I/O (PIO) and Direct Memory I/O (DMx). PIO is the same for all 50 Series processors. DMx for the earlier processors is like that of the 2350 to 2755 and 9650 to 9955 II with the exception that a few details of mapped I/O vary according to processor type. This variance is based on the IOTLB and the size of the cache as described below.

The IOTLB forms part of the virtual-to-physical address mapping hardware. (The STLB is the other part.) The IOTLB of all earlier processors contains 64 entries. Table B-13 shows the contents of each IOTLB entry for the earlier processors.

Table B-13
IOTLB Entry Format

Number of Bits			Contents	Description
750, 850	I450, 550-II	Rest of Earlier Processors		
12	12	12	Physical page number	Specifies a physical page in either of the I/O segments.
1	1	1	Valid bit	Indicates if this entry contains old data.
3	2	0	MBIO bits	Specifies the cache leaf to invalidate when writing to memory.

The earlier processors use Segment 0 only as an I/O segment. Each IOTLB entry for the earlier processors contains mapping information for one page of the I/O Segment 0 as shown below.

<u>IOTLB Entry</u>	<u>Corresponding Page in I/O Segments</u>
--------------------	---

0 to 63	Segment 0, Pages 0 to 63
---------	--------------------------

As noted in Table B-13, 0 to 3 bits of the virtual address form the MBIO bits for the earlier processors. As discussed in Chapter 11, these bits determine which part (leaf) of the cache to invalidate after a memory write. Three MBIO bits are used for an 8-leaf cache, 2 bits for a 4-leaf cache, and 0 bits for a 1-leaf cache.

Since the cache of the 750 and 850 contains 16K bytes, it contains mapping information about 8 entries of physical memory, each having the same page offset. The cache of the I450 and 550-II contains 8K bytes and contains mapping information about 8 entries of physical memory. The MBIO bits allow the information for only the modified entry to be invalidated after a memory write, rather than each of the 4 or 8 possible places.

The LIOT instruction loads the IOTLB entries with transfer information. For the 2350 to 9955 II as well as the earlier processors 850, 750, I450, and 550-II, the LIOT instruction must be used before any transfer occurs so that the processor maps virtual pages to the desired physical ones. The rest of the earlier processors load the IOTLB by accessing the appropriate page in segment 0 by an instruction, such as LDA, before any transfer since their cache is exactly the size of one page.

DMx transfer rates for the earlier processors are shown in Table B-14.

Table B-14
DMx Transfer Rates for the Earlier Processors

Type	Transfer	Maximum Speed
DMA	Input	2.5 Mbytes/sec
	Output	2.5 Mbytes/sec
DMC	Input	1.0 Mbytes/sec*
	Output	1.0 Mbytes/sec*
DMT	Input	2.5 Mbytes/sec*
	Output	2.5 Mbytes/sec*
DMQ	Input	280 Kbytes/sec*
	Output	280 Kbytes/sec*
Burst mode	Input	---
	Output	---

* This is an approximate value.

The format of a DMA control word in physical I/O mode is the same for all systems and is shown in Figure 11-4. For mapped I/O mode on the earlier systems, however, the format of the DMA control word is as shown in Figure B-7.

1	12 13	14 15	32
12's COMP WORD COUNT RESERVED OFFSET FOR START OF TRANSFER			

Control Word Format in Mapped I/O Mode
for the Earlier Processors

Figure B-7

All other aspects of DMx for the earlier processors are the same as those for the 2350 to 2755 and 9650 to 9955 II.

C

Process Exchange on the 850

Chapter 9 of this guide described process exchange for the single-stream members of the 50 Series family. On the dual-stream 850, however, process exchange is more complex because:

- There are two processing units, the ISUs
- Two processes can execute at once (one per ISU)
- The two ISUs share one set of PCBs, one ready list, and one set of wait lists

This chapter elaborates on each of these points. It also describes the elements of the 850 PXM, and describes the actions of the 850 dispatcher.

INSTRUCTION STREAM UNITS

Before reading this appendix, note the use of two terms. This ISU refers to the ISU on which a process of interest is currently executing. The Other ISU designates the second system ISU.

As mentioned in Chapter 1, the 850 contains two instruction stream units, or ISUs, each of which is equivalent to a 750 CPU. The ISUs operate independently of each other and are capable of performing any task any 750 processor can perform. The one exception is that only one ISU performs I/O and is thereby designated the master ISU. Therefore,

when a process running on the slave ISU wants to request I/O service, that process is moved to the master ISU for I/O service.

Two Executing Processes

Since there are two ISUs per system, two independent processes can be executing at the same time. These two processes are always the two having the highest level of priority in the entire system. Ensuring that the processes with the highest priority are the ones that are selected to execute makes dual-stream process exchange more complicated than its single-stream complement. It is further complicated by the fact that a process can be locked to one ISU, which means that it can only execute on a particular ISU (such as the backstop or supervisor). See the section The PX Lock, below, for more information about this topic.

One Set of Process Exchange Data Structures

To aid the ISUs in selecting the highest priority processes, the 850 uses one ready list, one group of wait lists, and one group of PCBs for both ISUs. This means that an ISU has to scan only one list to determine the processes available to execute. It also means the system has to maintain only one set of information, eliminating the need to check and update any duplicates. In addition, it means that a process not locked to one ISU may execute faster, since whichever ISU becomes available first can execute it.

850 PROCESS EXCHANGE ELEMENTS

The data structures of the 850 PXM include:

- PCBs
- Ready list
- Wait lists
- WAIT and NOTIFY instructions
- Dispatcher

Like its single-stream counterpart, the 850 PXM also manipulates the register file and the process interval timer. In addition, the 850 PXM uses the value CPUNUM and the PX lock to facilitate its operations.

The CPUNUM

CPUNUM is a 16-bit number stored in bits 1 to 16 of location '33 of the current register set. This number distinguishes the two ISUs. CPUNUM contains '41004 to represent This ISU and '102010 to represent The Other ISU.

The PX Lock

The PX lock ensures that only one ISU at a time has access to and can modify the contents of the process exchange data structures. This lock is a 16-bit number. When the lock contains 0, then either ISU can claim the right to access the structures. When it does not contain 0, the lock contains the same value as CPUNUM; that is, the id for one of the ISUs. Only the ISU specified by the lock can access the structures; the second ISU must wait until the first ISU is through its current task before gaining access.

PCBs

The process control block format for the 850 is nearly identical to that of the single-stream PCBs. Only a few locations contain added information, as shown in Table C-1.

OWNERH (bits 1 to 16 of location 25 in the current register set) specifies the segment containing all the PCBs. Each PCB contains at least 64 locations and must be aligned on a 128-byte boundary. The starting address of the PCB is also the process id.

No PCB (or any other data structure the PXM uses) should be contained in locations 0 to '37 of a segment. Each addressing mode handles address traps differently; avoiding these locations ensures that all addressing modes handle process exchange in the same way.

Table C-1
PCB Format for the 850

Section	Offset #	Contents
Control	0	Level pointer to BOL in ready list.
	1	Link pointer to next PCB, or 0.
	2 to 3	Segment #/offset of the semaphore on whose wait list this process currently resides. A segment # of 0 indicates that this PCB is on the ready list.
	4	Abort flags used to generate a process fault when this PCB is dispatched. Bits 1 to 15: Set by the software. Bit 16: Process interval timer overflow.
	5	Bits 1 to 4: Temporarily restrict process from running on one of the ISUs: 0000 = no restrictions 0100 = bar from This ISU 1000 = bar from The Other ISU Bit 5: Reserved for future use. Bits 6 to 7: If 01, this process last ran on This ISU; if 10, The Other ISU. Bit 8: If 0, the registers for this process have not been saved in the PCB. If 1, the registers have been saved in the PCB. Bits 9 to 11: Indicate which register set this process used last. Use the same format as the modals CRS field. Bit 12: Reserved for future use. Bits 13 to 16: Process is locked to: 0000 = neither ISU 0100 = This ISU 1000 = The Other ISU
	6 to 7	Reserved for future use.
Process State	'10 to '11	Process elapsed timers. This value is added to contents of PCB location '16 to give the number of msec this process has run. RTS can alter this location.
	'12 to '15	DTAR2 and DTAR3. These are never saved, only restored.

Table C-1 (continued)
PCB Format for the 850

Section	Offset #	Contents
	'16	Interval timer, bits 1 to 16.
	'17	Interval timer, bits 17 to 32.
	'20	Save mask. PXM uses this to avoid saving or restoring registers containing zeroes. Format of the word is: 1 to 8: GR0 to GR7 (8 32-bit registers) 9 to 12: FAC0 to FAC1 (4 32-bit registers) 13 to 16: base registers (4 32-bit registers (PB, SB, LB, XB)
	'21	Keys.
	'22 to '61	Storage for nonzero registers. (See Save mask, above.)
Fault	'62 to '63	Fault vector. Segment #/offset to fault table for Ring 0.
	'64 to '65	Fault vector. Segment #/offset to fault table for Ring 1.
	'66 to '67	Reserved for future use.
	'70 to '71	Fault vector. Segment #/offset to fault table for Ring 3.
	'72 to '73	Fault vector. Segment #/offset to fault table for page fault.
	'74 to '76	Concealed fault stack header (FIRST, NEXT, and LAST pointers).
	77	Reserved.
	'100 to 137	Concealed stack. These words can go anywhere in segment OWNERH; i.e., they do not have to start at location '100. The concealed stack can contain as many frames as desired.

Ready List and Wait Lists

The wait lists used in the 850 are identical to those found in the other 50 Series processors. The ready list is also identical except for the process exchange registers it uses.

Each ISU contains four process exchange registers. Two specify information about the currently running processes, and two specify information about the next processes to run. All four are 32 bits wide.

MY_PPA and OTHER_PPA define either the currently running process, or the process that is about to run. MY_PPA represents this process for This ISU; OTHER_PPA, for The Other ISU. Bits 1 to 16 of each register contain the process' level of priority; bits 17 to 32, the starting address of that process' PCB. Bits 1 to 16 of each register are guaranteed to always point to the ready list priority level that contains the highest priority process that is able to execute for the appropriate ISU.

The MY_PPNEXT register specifies the next process to run on This ISU; OTHER_PPNEXT, for The Other ISU. Like their single-stream counterpart (PPB), bits 1 to 16 specify the priority level of the next process to run, and bits 17 to 32 identify the PCB of this process. A nonzero value in bits 1 to 16 indicates valid contents.

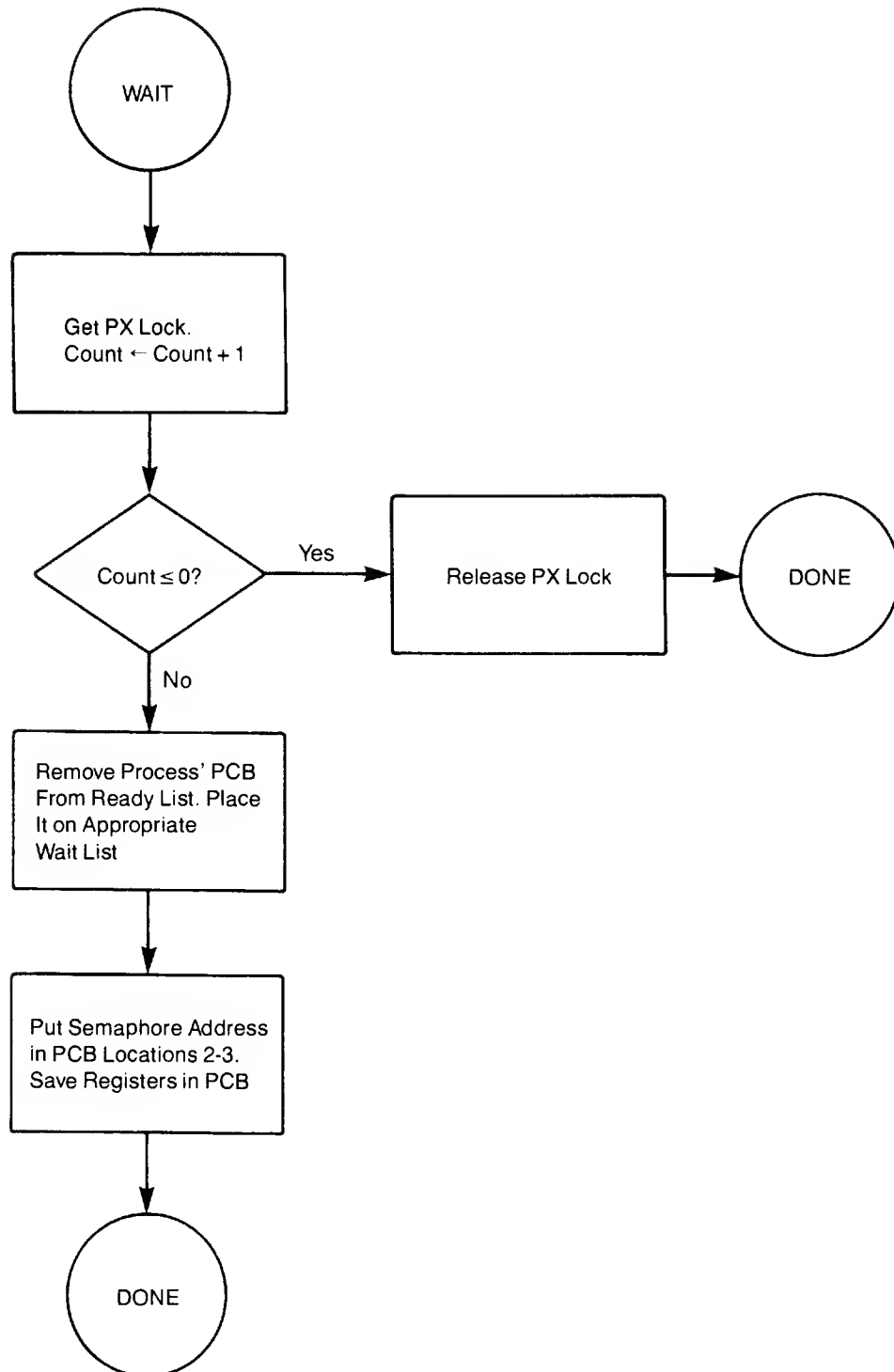
WAIT and NOTIFY Instructions

These instructions perform the same basic functions as their single-stream counterparts. However, their tasks also include obtaining the PX lock and loading the PXM registers with the correct information so that each ISU can correctly determine its own state and that of the second ISU. Figures C-1 and C-2, together with the text in this section, give simplified versions of how the 850 WAIT and NOTIFY instructions work.

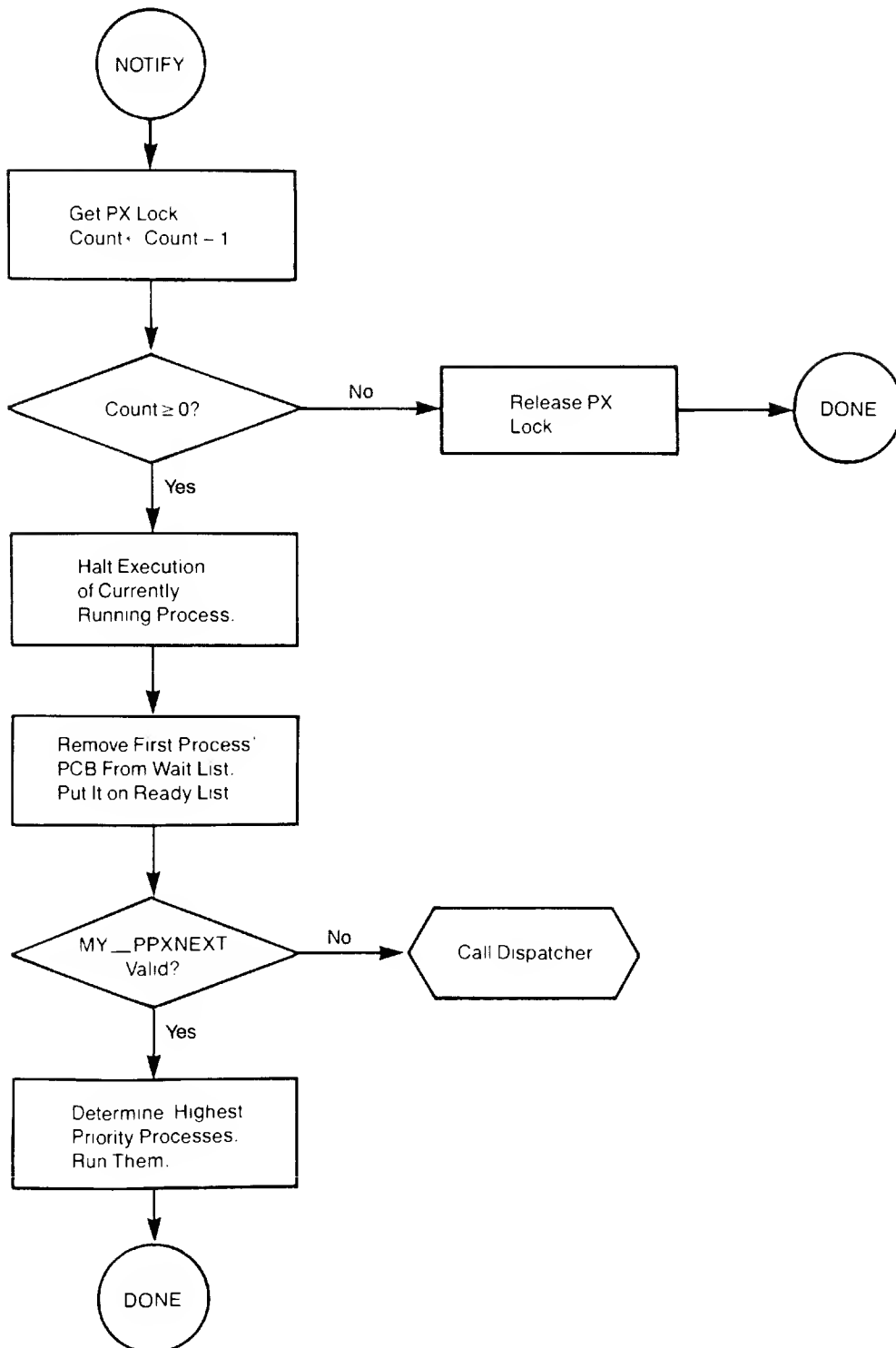
WAIT: WAIT tells the PXM to wait for an event to occur before executing more of the currently active process. The address pointer contained in WAIT specifies a semaphore on which the process is to wait. WAIT obtains the PX lock, then increments the semaphore count by 1.

If the incremented value is less than or equal to 0, WAIT releases the PX lock and performs no other actions. If the incremented value is greater than 0, WAIT removes the process' PCB from the ready list and places it on the appropriate wait list according to the process' priority. WAIT loads locations 2 and 3 of the process' PCB with the semaphore address and saves the process' base registers into its PCB.

After the short save, WAIT either runs the next process, if it knows it, or invokes the dispatcher to choose a new process to run.



The 850 WAIT Instruction
Figure C-1



The 850 NOTIFY Instruction
Figure C-2

NOTIFY: The 850 NOTIFY is significantly more complex than the single-stream WAIT. Its purpose is deceptively easy to state: NOTIFY ensures that the two currently running processes in the system are the two highest priority processes that are able to run. To do this, NOTIFY notifies the process that is at the top of the associated semaphore's wait list, then compares the priority level of this process with those of the two processes currently running.

Step 1. Finding a Process to Notify: When it executes a NOTIFY instruction, the PXM first acquires the PX lock. It then uses the pointer contained in the NOTIFY to reference a semaphore and decrement the semaphore count by 1. If the decremented value is less than 0, the PXM releases the PX lock and the NOTIFY is done.

If the decremented value is greater than or equal to 0, then the PXM must notify a process. It ceases to execute the current process and removes the first PCB on the semaphore's wait list. It places the PCB at the beginning or end of the appropriate level of the ready list as indicated by the NOTIFY.

Step 2. Choosing a Process to Run: The PXM must now choose a new process to run. If the contents of MY_PPNEXT are invalid, control transfers to the dispatcher, which determines the next process to run.

If the contents of MY_PPA are valid, the PXM must decide if the process it just notified is of higher priority than either of the processes currently executing. Six cases exist:

A < C and B < C
 C < B < A
 C < A < B
 C < A = B
 A < C < B
 B < C < A

where A is the process currently running on This ISU, B is the process currently running on The Other ISU, and C is the process that was just notified.

These cases can become quite involved, depending on where each of the three processes can run, and depending on what actions the PXM has taken previously. This discussion will explain two simple examples.

Suppose the first case were true. This means that C has the lowest priority of the three processes and will not be run. All the PXM needs to do is to decide on which ISU C is to be run.

If C can be run on only one ISU and has a higher priority than the process that ISU is to execute next (as specified in that ISU's MY_PPNEXT register), the PXM updates that ISU's MY_PPNEXT register so that it points to C. Therefore, that ISU will execute C next.

If C can be run on either ISU, the PXM updates MY_PPNEXT and OTHER_PPNEXT on both ISUs so that C will execute as soon as either ISU is free.

As another example, suppose case 2, $C < B < A$, were true. Here C has the highest priority of all, and should run on This ISU, if possible. A simplified algorithm for this case is shown in Figure C-3.

```

|
|   If C can run on This ISU
|   then if A can run on The Other ISU
|       then if OTHER_PPNEXT is of lower priority than A
|           then invalidate OTHER_PPNEXT
|               set MY_PPA to A
|   set PPA to C and go to the dispatcher.
|
|   If C can run on The Other ISU,
|   *** then if The Other ISU has received the most recent scan
|       then send this scan message. Scan identifies C as the
|           process to consider running;
|   else if priority of process in last scan is greater than C's
|       then return;
|       If priority of process in last scan is less than C's
|       then go to *** above;
|       If priority of process in last scan equals C's
|       then call dispatcher to scan ready list to
|           pick up the process queued first and return.
|
|
|-----

```

Sample NOTIFY Algorithm
Figure C-3

Dispatcher

Like its single-stream counterpart, the 850 dispatcher selects the next process to run and sets up the registers and conditions that process needs to run. The section DISPATCHER OPERATION, below, explains its actions.

Register Sets

Each ISU contains a register file identical to the single-stream register file. Each contains two user register sets designated as the current register set (CRS) and the other register set (ORS). Both of these have the same format as the user register sets on the single-stream processors.

Microsecond Timer

The 850 process timer is accurate to the microsecond. It is contained in two registers, `TIMERH` and `TIMERL`. `TIMERH` contains the two's complement of the millisecond portion of the clock. Bits 1 to 10 of `TIMERL` contain the microsecond part. Bits 11 to 16 of `TIMERL` are never changed.

Every 1.024 milliseconds the microsecond time overflows, causing a fetch cycle trap. The contents of `TIMERH` are incremented; upon overflow, bit 16 of the PCB abort flags is set to 1; a process fault occurs immediately and stops the current process from being executed.

DISPATCHER OPERATION

When a process completes execution or is aborted, the dispatcher begins to execute to select the next process to run. This discussion assumes that the PX lock contains the number of This ISU, so This ISU has the right to access the PXM data structures.

Step 1. Finding a Process to Run

The dispatcher first checks the contents of `MY_PPA`. If bits 17 to 32 are 0, the contents are invalid, as are the contents of `PPNEXT`. To find the next process to run, the dispatcher scans the ready list beginning at the level specified in bits 1 to 16 of `MY_PPA`.

The dispatcher scans the ready list until it finds the first process that is neither locked from This ISU, nor currently running on The Other ISU. Any processes the dispatcher finds during the scan that are temporarily locked from This ISU are unlocked by setting the lock field in the process' PCB location 5 to 0. If the ready list contains no suitable process, the dispatcher releases the PX lock.

If the dispatcher finds a process on the ready list to run, it next checks for two things:

- Does the `OTHER_PPNEXT` point to this process?
- Has This ISU sent a scan message to The Other ISU suggesting that The Other ISU run this process?

If the `OTHER_PPNEXT` points to this process, This ISU will not run this process. It will be run at a later date on The Other ISU.

If a scan message was sent, the dispatcher invalidates the message so that The Other ISU will not run this process. Once the message is invalidated, or if no such message was sent, the dispatcher loads `MY_PPA` with the level and PCB starting address of this process.

Step 2. Locating Register Values and a Register Set

Once MY_PPA contains valid information, the dispatcher must locate the register values this process needs for execution, and must find a register set to contain them. The values can be in one of three places:

- In a register set on This ISU
- In a register set on the Other ISU
- In the process' PCB

The dispatcher checks the CRS field in the process' PCB to see if either of This ISU's register sets or either of The Other ISU's register sets already contain the process' register values. If either of This ISU's register sets do, the dispatcher makes that set the CRS.

If either of The Other ISU's register sets contain the process' values, the dispatcher sends a message to The Other ISU telling it to save the contents of that register set into the process' PCB. The dispatcher then releases the PX lock so that The Other ISU can save the values. After a short time, This ISU regains the PX lock and tries to choose a register set from the beginning.

If none of the register sets on either ISU already contains the process' register values, the dispatcher must load them from the process' PCB. The dispatcher chooses a register set on This ISU by checking the Save Done bit of both the CRS and ORS.

If the Save Done bit of the CRS contains a 1, the CRS is available. The dispatcher loads the process' values from the PCB into the CRS.

If the CRS is not available, the dispatcher checks the Save Done bit of the ORS. If the Save Done bit contains a 1, the dispatcher makes ORS the CRS, then loads in the process' register values.

If neither the CRS nor the ORS is available (both Save Done bits contain 0), the dispatcher saves the contents of the ORS into the appropriate PCB, makes ORS the CRS, then loads the process' register values into it.

Step 3. Updating Information and Running the New Process

After choosing and loading (if necessary) a register set, the dispatcher loads location 5 of the process' PCB with the id of the ISU on which the process most recently ran. It also loads the PCB with the location of the process' register values, and sets bits 15 and 16 of the keys to 0. The dispatcher then releases the PX lock and enables the microsecond timer. The new process begins to execute.

D

Instruction Summary Charts

This appendix contains two instruction summary charts: one for S mode, R mode, and V mode; another for I mode. Each chart contains a list of instructions for the Prime 50 Series processors. (Appendix D lists those instructions that have been archived.) Each instruction is followed by its octal code, format, function, addressing mode, CBIT, LINK, and condition code information, and a one-line description of the instruction.

The columns in each chart are as follows:

R Restrictions:

Blank	Regular instruction.
R	Instruction causes a restricted mode fault if executed in other than Ring 0.
P	Instruction may cause a fault depending on address.

Mnem A mnemonic name recognized by the assembler PMA.

Opcode Octal operation code portion of the instruction.

RI Register (R) and Immediate (I) forms, if available.

SYSTEM ARCHITECTURE REFERENCE GUIDE

Form Format of instruction:

<u>Mnemonic</u>	<u>Definition</u>
AP	Address Pointer
BRAN	Branch
CHAR	Character
DECI	Decimal
GEN	Generic
GR	General Register -- non Memory Reference
IBRN	I Mode Branch
MR	Memory Reference -- Non I Mode
MRFR	Memory Reference -- Floating Register
MRGR	Memory Reference -- General Register
MRNR	Memory Reference -- Non Register
PIO	Programmed I/O
RGEN	Register Generic
SHFT	Shift

Func Function of instruction:

<u>Mnemonic</u>	<u>Definition</u>
ADMOD	Addressing Mode
BRAN	Branch
CHAR	Character
CLEAR	Clear Field
CPTR	C Language Pointer
DECI	Decimal Arithmetic
FIELD	Field Register
FLPT	Floating Point Arithmetic
GRR	General Register Relative
INT	Integer
INTGY	Integrity
IO	Input/Output
KEYS	Keys
LOGIC	Logical Operations
LTSTS	Logical Test and Set
MCTL	Machine Control
MOVE	Move
PCTLJ	Program Control and Jump
PRCEX	Process Exchange
QUEUE	Queue Control
SHIFT	Register Shift
SKIP	Skip

M Addressing modes of instructions:

<u>Mode</u>	<u>Name</u>
S	Sectored
R	Relative
V	Virtual (64V)
I	32I

C How instruction affects the CBIT and LINK.

<u>Code</u>	<u>Definition</u>
-	CBIT and LINK are unchanged
1	CBIT = unchanged; LINK = carry
2	CBIT = overflow status; LINK = carry
3	CBIT = overflow status; LINK = indeterminate
4	CBIT = shift extension; LINK = shift extension
5	CBIT = result; LINK = indeterminate
6	CBIT and LINK are indeterminate
7	CBIT and LINK are loaded by the instruction
8	CBIT = result; LINK = unchanged
9	CBIT = unchanged; LINK = indeterminate
*	CBIT and LINK values vary among processors; see individual instruction description

CC How instruction affects the condition codes.

<u>Code</u>	<u>Definition</u>
-	Condition codes are unchanged.
1	Condition codes are set to reflect the result of arithmetic operation or compare.
4	Condition codes are set to reflect result of branch, compare, or logicize operand state.
5	Condition codes are indeterminate.
6	Condition codes are loaded by instruction.
7	Condition codes show special results for this instruction.

Description A brief description of the instruction.

Table D-1 contains a summary of S mode, R mode, and V mode instructions. Table D-2 is a summary of I mode instructions. Instructions that have been archived are not in either of these tables; see Appendix D for them.

Table D-1
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	A1A	141206	GEN	INT	SRV	2	1	Add One to A
	A2A	140304	GEN	INT	SRV	2	1	Add Two to A
	ABQ	141716	AP	QUEUE	V	-	7	Add Entry to Bottom of Queue
	ACA	141216	GEN	INT	SRV	2	1	Add CBIT to A
	ADD	06	MR	INT	SRV	2	1	Add
	ADL	06 03	MR	INT	V	2	1	Add Long
	ADLL	141000	GEN	INT	V	2	1	Add LINK to L
	ALFA 0	001301	GEN	FIELD	V	6	-	Add L to FAR 0
	ALFA 1	001311	GEN	FIELD	V	6	-	Add L to FAR 1
	ALL	0414XX	SHFT	SHIFT	SRV	4	-	A Left Logical
	ALR	0416XX	SHFT	SHIFT	SRV	4	-	A Left Rotate
	ALS	0415XX	SHFT	SHIFT	SRV	3	-	A Arithmetic Left Shift
	ANA	03	MR	LOGIC	SRV	-	-	AND to A
	ANL	03 03	MR	LOGIC	V	-	-	AND to A Long
	ARGT	000605	GEN	PCTLJ	V	6	5	Argument Transfer
	ARL	0404XX	SHFT	SHIFT	SRV	4	-	A Right Logical
	ARR	0406XX	SHFT	SHIFT	SRV	4	-	A Right Rotate
	ARS	0405XX	SHFT	SHIFT	SRV	4	-	A Arithmetic Right Shift
	ATQ	141717	AP	QUEUE	V	-	7	Add Entry to Top of Queue
	BCEQ	141602	BRAN	BRAN	V	-	-	Branch on Condition Code EQ
	BOGE	141605	BRAN	BRAN	V	-	-	Branch on Condition Code GE
	BOGT	141601	BRAN	BRAN	V	-	-	Branch on Condition Code GT
	BCLE	141600	BRAN	BRAN	V	-	-	Branch on Condition Code LE
	BCLT	141604	BRAN	BRAN	V	-	-	Branch on Condition Code LT
	BCNE	141603	BRAN	BRAN	V	-	-	Branch on Condition Code NE
	BCR	141705	BRAN	BRAN	V	-	-	Branch on CBIT Reset to 0
	BCS	141704	BRAN	BRAN	V	-	-	Branch on CBIT Set to 1
	BDX	140734	BRAN	BRAN	V	-	-	Branch on Decrement X
	BDY	140724	BRAN	BRAN	V	-	-	Branch on Decrement Y
	BEQ	140612	BRAN	BRAN	V	-	4	Branch on A Equal to 0
	BFEQ	141612	BRAN	BRAN	V	-	4	Branch on F Equal to 0
	BFGE	141615	BRAN	BRAN	V	-	4	Branch on F Greater Than or Equal to 0
	BFGT	141611	BRAN	BRAN	V	-	4	Branch on F Greater Than 0
	BFLE	141610	BRAN	BRAN	V	-	4	Branch on F Less Than or Equal to 0
	BFLT	141614	BRAN	BRAN	V	-	4	Branch on F Less Than 0
	BFNE	141613	BRAN	BRAN	V	-	4	Branch on F Not Equal to 0
	BGE	140615	BRAN	BRAN	V	-	4	Branch on A Greater Than or Equal to 0
	BGT	140611	BRAN	BRAN	V	-	4	Branch on A Greater Than 0
	BIX	141334	BRAN	BRAN	V	-	-	Branch on Incremented X
	BIY	141324	BRAN	BRAN	V	-	-	Branch on Incremented Y
	BLE	140610	BRAN	BRAN	V	-	4	Branch on A Less Than or Equal to 0
	BLEQ	140702	BRAN	BRAN	V	-	4	Branch on L Equal to 0

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	BLGE	140615	BRAN	BRAN	V	-	4	Branch on L Greater Than or Equal to 0
	BLGT	140701	BRAN	BRAN	V	-	4	Branch on L Greater Than 0
	BLLE	140700	BRAN	BRAN	V	-	4	Branch on L Less Than or Equal to 0
	BLLT	140614	BRAN	BRAN	V	-	4	Branch on L Less Than 0
	BLNE	140703	BRAN	BRAN	V	-	4	Branch on L Not Equal to 0
	BLR	141707	BRAN	BRAN	V	-	-	Branch on LINK Reset to 0
	BLS	141706	BRAN	BRAN	V	-	-	Branch on LINK Set to 1
	BLT	140614	BRAN	BRAN	V	-	4	Branch on A Less Than 0
	BMEQ	141602	BRAN	BRAN	V	-	-	Branch on Magnitude Condition EQ
	BMGE	141706	BRAN	BRAN	V	-	-	Branch on Magnitude Condition GE
	BMGT	141710	BRAN	BRAN	V	-	-	Branch on Magnitude Condition GT
	BMLE	141711	BRAN	BRAN	V	-	-	Branch on Magnitude Condition LE
	BMLT	141707	BRAN	BRAN	V	-	-	Branch on Magnitude Condition LT
	BMNE	141603	BRAN	BRAN	V	-	-	Branch on Magnitude Condition NE
	BNE	140613	BRAN	BRAN	V	-	4	Branch on A Not Equal to 0
	CAL	141050	GEN	CLEAR	SRV	-	-	Clear A Left Byte
	CALF	000705	AP	PCTLJ	V	6	5	Call Fault Handler
	CAR	141044	GEN	CLEAR	SRV	-	-	Clear A Right Byte
	CAS	11	MR	SKIP	SRV	1	1	Compare A and Skip
	CAZ	140214	GEN	SKIP	SRV	1	1	Compare A with 0
	CEA	000111	GEN	PCTLJ	SR	-	-	Compute Effective Address
	OGT	001314	GEN	BRAN	V	6	5	Computed GOTO
	CHS	140024	GEN	INT	SRV	-	-	Change Sign
	CLS	11 03	MR	LOGIC	V	1	1	Compare L and Skip
	CMA	140401	GEN	LOGIC	SRV	-	-	Complement A
	CRA	140040	GEN	CLEAR	SRV	-	-	Clear A to 0
	CRB	140015	GEN	CLEAR	SRV	-	-	Clear B to 0
	CRE	141404	GEN	CLEAR	V	-	-	Clear E to 0
	CRL	140010	GEN	CLEAR	SRV	-	-	Clear L to 0
	CRLE	141410	GEN	CLEAR	V	-	-	Clear L and E to 0
	CSA	140320	GEN	MOVE	SRV	5	-	Copy Sign of A
	DAD	06	MR	INT	SR	2	1	Double Add
	DBL	000007	GEN	INT	SR	-	-	Enter Double Precision Mode
	DFAD	06 02	MR	FLPT	RV	3	5	Double Precision Floating Add
	DFCM	140574	GEN	FLPT	RV	3	5	Double Precision Floating Complement
	DFCS	11 02	MR	FLPT	RV	6	5	Double Precision Floating Compare and Skip

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	DFDV	17 02	MR	FLPT	RV	3	5	Double Precision Floating Divide
	DFLD	02 02	MR	FLPT	RV	-	-	Double Precision Floating Load
	DFLX	15 02	MR	FLPT	V	-	-	Double Precision Floating Load Index
	DFMP	16 02	MR	FLPT	RV	3	5	Double Precision Floating Multiply
	DFSB	07 02	MR	FLPT	RV	3	5	Double Precision Floating Subtract
	DFST	04 02	MR	FLPT	RV	-	-	Double Precision Floating Store
	DIV	17	MR	INT	V	3	5	Divide
	DIV	17	MR	INT	SR	3	5	Divide
	DLD	02	MR	MOVE	SR	-	-	Double Load
	DRN	040300	GEN	FLPT	V	3	5	Double Round From Quad
	DRNM	140571	GEN	FLPT	V	8	5	Double Round From Quad Towards Negative Infinity
	DRNP	040301	GEN	FLPT	V	3	5	Double Round From Quad Towards Positive Infinity
	DRNZ	040302	GEN	FLPT	V	3	5	Double Round From Quad Towards Zero
	DRX	140210	GEN	SKIP	SRV	-	-	Decrement and Replace X
	DSB	07	MR	INT	SR	2	1	Double Subtract
	DST	04	MR	MOVE	SR	-	-	Double Store
	DVL	17 03	MR	INT	V	3	5	Divide Long
	E16S	000011	GEN	ADMOD	SRV	-	-	Enter 16S Mode
	E32I	001010	GEN	ADMOD	SRV	-	-	Enter 32I Mode
	E32R	001013	GEN	ADMOD	SRV	-	-	Enter 32R Mode
	E32S	000013	GEN	ADMOD	SRV	-	-	Enter 32S Mode
	E64R	001011	GEN	ADMOD	SRV	-	-	Enter 64R Mode
	E64V	000010	GEN	ADMOD	SRV	-	-	Enter 64V Mode
	EAA	01 01	MR	MOVE	R	-	-	Effective Address to A
	EAFA 0	001300	AP	FIELD	V	-	-	Effective Address to FAR 0
	EAFA 1	001310	AP	FIELD	V	-	-	Effective Address to FAR 1
	EAL	01 01	MR	PCTLJ	V	-	-	Effective Address to L
	EALB	13 02	MR	PCTLJ	V	-	-	Effective Address to LB
	EAXB	12 02	MR	PCTLJ	V	-	-	Effective Address to XB
R	EIO	14 01	MR	IO	V	-	7	Execute I/O
R	ENB	000401	GEN	IO	SRV	-	-	Enable Interrupts
R	ENBL	000401	GEN	IO	SRV	-	-	Enable Interrupts (Local)
R	ENBM	000400	GEN	IO	SRV	-	-	Enable Interrupts (Mutual)
R	ENBP	000402	GEN	IO	SRV	-	-	Enable Interrupts (Process)
	ERA	05	MR	LOGIC	SRV	-	-	Exclusive OR to A
	ERL	05 03	MR	LOGIC	V	-	-	Exclusive OR to L
	FAD	06 01	MR	FLPT	RV	3	5	Floating Add

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	OC	Description
	FCDQ	140571	GEN	FLPT	V	-	-	Floating Convert Double to Quad
	FCM	140530	GEN	FLPT	RV	3	5	Floating Complement
	FCS	11 01	MR	FLPT	RV	6	5	Floating Compare and Skip
	FDBL	140016	GEN	FLPT	V	-	-	Floating Convert Single to Double
	FDV	17 01	MR	FLPT	RV	3	5	Floating Divide
	FLD	02 01	MR	FLPT	RV	-	-	Floating Load
	FLOT	140550	GEN	FLPT	R	6	5	Convert Integer to Floating Point
	FLTA	140532	GEN	FLPT	V	6	5	Convert Integer to Floating Point
	FLTL	140535	GEN	FLPT	V	6	5	Convert Long Integer to Floating Point
	FLX	15 01	MR	FLPT	RV	-	-	Floating Load Index
	FMP	16 01	MR	FLPT	RV	3	5	Floating Multiply
	FRN	140534	GEN	FLPT	RV	3	5	Floating Round
	FRNM	040320	GEN	FLPT	V	3	5	Floating Round Towards Negative Infinity
	FRNP	040303	GEN	FLPT	V	3	5	Floating Round Towards Positive Infinity
	FRNZ	040321	GEN	FLPT	V	3	5	Floating Round Towards Zero
	FSB	07 01	MR	FLPT	RV	3	5	Floating Subtract
	FSGT	140515	GEN	FLPT	RV	-	5	Floating Skip If Greater Than 0
	FSLE	140514	GEN	FLPT	RV	-	5	Floating Skip If Less Than or Equal to 0
	FSMI	140512	GEN	FLPT	RV	-	5	Floating Skip If Minus
	FSNZ	140511	GEN	FLPT	RV	-	5	Floating Skip If Not Equal to 0
	FSPL	140513	GEN	FLPT	RV	-	5	Floating Skip If Plus
	FST	04 01	MR	FLPT	RV	3	5	Floating Store
	FSZE	140510	GEN	FLPT	RV	-	5	Floating Skip If Equal to 0
R	HLT	000000	GEN	MCTL	SRV	-	-	Halt
	IAB	000201	GEN	MOVE	SRV	-	-	Interchange A and B
	ICA	141340	GEN	MOVE	SRV	-	-	Interchange Bytes of A
	ICL	141140	GEN	MOVE	SRV	-	-	Interchange Bytes and Clear Left
	ICR	141240	GEN	MOVE	SRV	-	-	Interchange Bytes and Clear Right
	ILE	141414	GEN	MOVE	V	-	-	Interchange L and E
	IMA	13	MR	MOVE	SRV	-	-	Interchange Memory and A
R	INA	54	PIO	IO	SR	-	-	Input to A
R	INBC	001217	AP	PRCEX	V	6	5	Interrupt Notify Beginning, Clear Active Interrupt
R	INEN	001215	AP	PRCEX	V	6	5	Interrupt Notify Beginning

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
R	INEC	001216	AP	PRCEX	V	6	5	Interrupt Notify End, Clear Active Interrupt
R	INEN	001214	AP	PRCEX	V	6	5	Interrupt Notify End
R	INH	001001	GEN	IO	SRV	-	-	Inhibit Interrupts
R	INHL	001001	GEN	IO	SRV	-	-	Inhibit Interrupts (Local)
R	INHM	001000	GEN	IO	SRV	-	-	Inhibit Interrupts (Mutual)
R	INHPT	001002	GEN	IO	SRV	-	-	Inhibit Interrupts (Process)
	INK	000043	GEN	KEYS	SR	-	-	Input Keys
	INT	140554	GEN	FLPT	R	3	5	Convert Floating Point to Integer
	INTA	140531	GEN	FLPT	V	3	5	Convert Floating Point to Integer
	INTL	140533	GEN	FLPT	V	3	5	Convert Floating Point to Integer Long
	IRS	12	MR	SKIP	SRV	-	-	Increment and Replace Memory
R	IRTC	000603	GEN	IO	V	7	6	Interrupt Return, Clear Active Interrupt
R	IRTN	000601	GEN	IO	V	7	6	Interrupt Return
	IRX	140114	GEN	SKIP	SRV	-	-	Increment and Replace X
R	ITLB	000615	GEN	MCIL	V	6	5	Invalidate STLB Entry
	JDX	15 02	MR	PCILJ	R	-	-	Jump and Decrement X
	JIX	15 03	MR	PCILJ	R	-	-	Jump and Increment X
	JMP	01	MR	PCILJ	SRV	-	-	Jump
	JST	10	MR	PCILJ	SRV	-	-	Jump and Store
	JSX	35 03	MR	PCILJ	RV	-	-	Jump and Save in X
	JSXB	14 02	MR	PCILJ	V	-	-	Jump and Save in XB
	JSY	14	MR	PCILJ	V	-	-	Jump and Save in Y
	LCEQ	141503	GEN	LTSTS	V	-	-	Load A on Condition Code EQ
	LOGE	141504	GEN	LTSTS	V	-	-	Load A on Condition Code GE
	LOGT	141505	GEN	LTSTS	V	-	-	Load A on Condition Code GT
	LCLE	141501	GEN	LTSTS	V	-	-	Load A on Condition Code LE
	LCLT	141500	GEN	LTSTS	V	-	-	Load A on Condition Code LT
	LCNE	141502	GEN	LTSTS	V	-	-	Load A on Condition Code NE
	LDA	02	MR	MOVE	SRV	-	-	Load A
	LDC 0	001302	CHAR	CHAR	V	-	7	Load Character
	LDC 1	001312	CHAR	CHAR	V	-	7	Load Character
	LDL	02 03	MR	MOVE	V	-	-	Load Long
P	LDLR	05 01	MR	MOVE	V	-	5	Load from Addressed Register
	LDX	35 00	MR	MOVE	SRV	-	-	Load X
	LDY	35 01	MR	MOVE	V	-	-	Load Y
	LEQ	140413	GEN	LTSTS	SRV	-	4	Load A on A Equal to 0
	LF	140416	GEN	LTSTS	SRV	-	5	Load False
	LFEQ	141113	GEN	LTSTS	V	-	4	Load A on F Equal to 0
	LFGE	141114	GEN	LTSTS	V	-	4	Load A on F Greater Than or Equal to 0
	LFGT	141115	GEN	LTSTS	V	-	4	Load A on F Greater Than 0

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	LFILE	141111	GEN	LTSTS	V	-	4	Load A on F Less Than or Equal to 0
	LFLI 0	001303	BRAN	FIELD	V	-	-	Load FIR 0 Immediate
	LFLI 1	001313	BRAN	FIELD	V	-	-	Load FIR 1 Immediate
	LFLT	141110	GEN	LTSTS	V	-	4	Load A on F Less Than 0
	LFNE	141112	GEN	LTSTS	V	-	4	Load A on F Not Equal to 0
	LGE	140414	GEN	LTSTS	SRV	-	4	Load A on A Greater Than or Equal to 0
	LGT	140415	GEN	LTSTS	SRV	-	4	Load A on A Greater Than 0
R	LIOT	000044	AP	MCTL	V	6	5	Load IOTLB
	LLE	140411	GEN	LTSTS	SRV	-	4	Load A on A Less Than or Equal to 0
	LLEQ	141513	GEN	LTSTS	V	-	4	Load L on A Equal to 0
	LLGE	140414	GEN	LTSTS	V	-	4	Load L on A Greater Than or Equal to 0
	LLGT	141515	GEN	LTSTS	V	-	4	Load L on A Greater Than 0
	LLL	0410XX	SHFT	SHIFT	SRV	4	-	Long Left Logical
	LLLE	141511	GEN	LTSTS	V	-	4	Load L on A Less Than or Equal to 0
	LLLT	140410	GEN	LTSTS	V	-	4	Load L on A Less Than 0
	LLNE	141512	GEN	LTSTS	V	-	4	Load L on A Not Equal to 0
	LLR	0412XX	SHFT	SHIFT	SRV	4	-	Long Left Rotate
	LLS	0411XX	SHFT	SHIFT	SRV	3	5	Long Left Shift
	LLT	140410	GEN	LTSTS	SRV	-	4	Load A on A Less Than 0
	LINE	140412	GEN	LTSTS	SRV	-	4	Load A on A Not Equal to 0
R	LPID	000617	GEN	MCTL	V	-	-	Load Process ID
R	LPSW	000711	AP	MCTL	V	7	6	Load Process Status Word
	LRL	0400XX	SHFT	SHIFT	SRV	4	-	Long Right Logical
	LRR	0402XX	SHFT	SHIFT	SRV	4	-	Long Right Rotate
	LRS	0401XX	SHFT	SHIFT	SRV	4	-	Long Right Shift
	LT	140417	GEN	LTSTS	SRV	-	5	Load True
	MPL	16 03	MR	INT	V	*	-	Multiply Long
	MPY	16	MR	INT	V	3	-	Multiply
	MPY	16	MR	INT	SR	3	*	Multiply
R	NFYB	001211	AP	PRCEX	V	6	5	Notify
R	NFYE	001210	AP	PRCEX	V	6	5	Notify
	NOP	000001	GEN	MCTL	SRV	-	-	No Operation
R	OCP	14	PIO	IO	SR	-	-	Output Control Pulse
	ORA	03 02	MR	LOGIC	V	-	-	Inclusive OR
R	OTA	74	PIO	IO	SR	-	-	Output from A
	OTK	000405	GEN	KEYS	SR	7	6	Output Keys
	PCL	10 02	MR	PCTLJ	V	6	5	Procedure Call
	PID	000211	GEN	INT	SR	-	-	Position for Integer Divide
	PIDA	000115	GEN	INT	V	-	-	Position for Integer Divide
	PIDL	000305	GEN	INT	V	-	-	Position for Integer Divide Long
	PIM	000205	GEN	INT	SR	-	-	Position after Multiply

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	PIMA	000015	GEN	INT	V	3	5	Position after Multiply
	PIML	000301	GEN	INT	V	3	5	Position after Multiply Long
	PRIN	000611	GEN	PCTLJ	V	7	6	Procedure Return
R	PILB	000064	GEN	MCTL	V	6	5	Purge TLB
	QFAD	5 2 2	MR	FLPT	V	3	5	Quad Precision Floating Add
	QFCM	140570	GEN	FLPT	V	3	5	Quad Precision Floating Complement
	QFCS	5 2 6	MR	FLPT	V	6	5	Quad Precision Floating Compare and Skip
	QFDV	5 2 5	MR	FLPT	V	3	5	Quad Precision Floating Divide
	QFLD	5 2 0	MR	FLPT	V	-	-	Quad Precision Floating Load
	QFLX	6 7	MR	FLPT	V	-	-	Quad Precision Floating Load Index
	QFMP	5 2 4	MR	FLPT	V	3	5	Quad Precision Floating Multiply
	QFSB	5 2 3	MR	FLPT	V	3	5	Quad Precision Floating Subtract
	QFST	5 2 1	MR	FLPT	V	-	-	Quad Precision Floating Store
	QINQ	140572	GEN	FLPT	V	3	5	Quad to Integer, in Quad Convert
	QIQR	140573	GEN	FLPT	V	3	5	Quad to Integer, in Quad Convert Rounded
	RBQ	141715	AP	QUEUE	V	-	7	Remove Entry from Bottom of Queue
	RCB	140200	GEN	KEYS	SRV	8	-	Reset CBIT to 0
R	RMC	000021	GEN	INTGY	SRV	-	-	Reset Machine Check Flag to 0
	RRST	000717	AP	MCTL	V	-	-	Restore Registers
	RSAB	000715	AP	MCTL	V	-	-	Save Registers
	RTQ	141714	AP	QUEUE	V	-	7	Remove Entry from Top of Queue
R	RTS	000511	GEN	MCTL	V	-	-	Reset Time Slice
	S1A	140110	GEN	INT	SRV	2	1	Subtract 1 from A
	S2A	140310	GEN	INT	SRV	2	1	Subtract 2 from A
	SAR	10026X	GEN	SKIP	SRV	-	-	Skip on A Register Bit Reset to 0
	SAS	10126X	GEN	SKIP	SRV	-	-	Skip on A Register Bit Set to 1
	SBL	07 03	MR	INT	V	2	1	Subtract Long
	SCB	140600	GEN	KEYS	SRV	5	-	Set CBIT to 1
	SGL	000005	GEN	INT	SR	-	-	Enter Single Precision Mode
	SGT	100220	GEN	SKIP	SRV	-	-	Skip on A Greater Than 0
	SKP	100000	GEN	SKIP	SRV	-	-	Skip
R	SKS	34	PIO	IO	SR	-	-	Skip on Condition Satisfied

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	SLE	101220	GEN	SKIP	SRV	-	-	Skip on A Less Than or Equal to 0
	SLN	101100	GEN	SKIP	SRV	-	-	Skip on LSB of A Nonzero
	SLZ	100100	GEN	SKIP	SRV	-	-	Skip on LSB of A Zero
	SMCR	100200	GEN	INTGY	SRV	-	-	Skip on Machine Check Reset to 0
	SMCS	101200	GEN	INTGY	SRV	-	-	Skip on Machine Check Set to 1
	SMI	101400	GEN	SKIP	SRV	-	-	Skip on A Minus
	SNZ	101040	GEN	SKIP	SRV	-	-	Skip on A Nonzero
	SPL	100400	GEN	SKIP	SRV	-	-	Skip on A Plus
	SRC	100001	GEN	SKIP	SRV	-	-	Skip on CBIT Reset to 0
	SSC	101001	GEN	SKIP	SRV	-	-	Skip on CBIT Set to 1
	SSM	140500	GEN	INT	SRV	-	-	Set Sign of A Minus
	SSP	140100	GEN	INT	SRV	-	-	Set Sign of A Plus
	SSSN	040310	GEN	MCTL	V	6	5	Store System Serial Number
	STA	04	MR	MOVE	SRV	-	-	Store A into Memory
	STAC	001200	AP	MOVE	V	-	7	Store A Conditionally
	STC 0	001322	CHAR	CHAR	V	-	7	Store Character
	STC 1	001332	CHAR	CHAR	V	-	7	Store Character
	STEX	001315	GEN	PCTLJ	V	6	5	Stack Extend
	STFA 0	001320	AP	FIELD	V	-	-	Store FAR 0
	STFA 1	001330	AP	FIELD	V	-	-	Store FAR 1
	STL	04 03	MR	MOVE	V	-	-	Store Long
	STLC	001204	AP	MOVE	V	-	7	Store L Conditionally
P	STLR	03 01	MR	MOVE	V	-	5	Store L into Addressed Register
R	STPM	000024	GEN	MCTL	V	-	-	Store Processor Model Number
	STPM	000510	GEN	MCTL	V	6	5	Store Process Timer
	STX	15	MR	MOVE	SRV	-	-	Store X
	STY	35 02	MR	MOVE	V	-	-	Store Y
	SUB	07	MR	INT	SRV	2	1	Subtract
	SVC	000505	GEN	PCTLJ	SRV	-	-	Supervisor Call
	SZE	100040	GEN	SKIP	SRV	-	-	Skip on A Zero
	TAB	140314	GEN	MOVE	V	-	-	Transfer A to B
	TAK	001015	GEN	KEYS	V	7	6	Transfer A to Keys
	TAX	140504	GEN	MOVE	V	-	-	Transfer A to X
	TAY	140505	GEN	MOVE	V	-	-	Transfer A to Y
	TBA	140604	GEN	MOVE	V	-	-	Transfer B to A
	TCA	140407	GEN	INT	SRV	2	1	Two's Complement A
	TCL	141210	GEN	INT	V	2	1	Two's Complement Long
	TFLL 0	001323	GEN	FIELD	V	-	-	Transfer FLR 0 to L
	TFLL 1	001333	GEN	FIELD	V	-	-	Transfer FLR 1 to L
	TKA	001005	GEN	KEYS	V	-	-	Transfer Keys to A
	TLFL 0	001321	GEN	FIELD	V	-	-	Transfer L to FLR 0
	TLFL 1	001331	GEN	FIELD	V	-	-	Transfer L to FLR 1
	TSTQ	141757	AP	QUEUE	V	-	7	Test Queue

Table D-1 (continued)
S Mode, R Mode, and V Mode Instruction Summary

R	Mnem	Opcode	Form	Func	M	C	CC	Description
	TXA	141034	GEN	MOVE	V	-	-	Transfer X to A
	TYA	141124	GEN	MOVE	V	-	-	Transfer Y to A
R	WAIT	000315	AP	PRCEX	V	-	-	Wait
	XAD	001100	DECI	DECI	V	3	1	Decimal Add
	XBTD	001145	DECI	DECI	V	3	5	Binary to Decimal Conversion
	XCA	140104	GEN	MOVE	SRV	-	-	Exchange and Clear A
	XCB	140204	GEN	MOVE	SRV	-	-	Exchange and Clear B
	XCM	001102	DECI	DECI	V	-	1	Decimal Compare
	XDTB	001146	DECI	DECI	V	3	5	Decimal to Binary Conversion
	XDV	001107	DECI	DECI	V	3	5	Decimal Divide
	XEC	01 02	MR	PCTLJ	RV	-	-	Execute
	XED	001112	DECI	DECI	V	-	-	Numeric Edit
	XMP	001104	DECI	DECI	V	3	1	Decimal Multiply
	XMV	001101	DECI	DECI	V	3	1	Decimal Move
	ZCM	001117	CHAR	CHAR	V	6	7	Compare Character Field
	ZED	001111	CHAR	CHAR	V	-	-	Character Field Edit
	ZFIL	001116	CHAR	CHAR	V	6	5	Fill Field With Character
	ZMV	001114	CHAR	CHAR	V	6	5	Move Character Field
	ZMVD	001115	CHAR	CHAR	V	6	5	Move Characters Between Equal Length Strings
	ZTRN	001110	CHAR	CHAR	V	-	-	Character String Translate

Table D-2
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	A	02	RI	MRGR	INT	2	1	Add Fullword
	ABQ	134		AP	QUEUE	-	7	Add Entry to Bottom of Queue
	ACP	55	RI	GR	CPTR	-	-	Add C Pointer
	ADLR	014		RGEN	INT	2	1	Add LINK to R
	AH	12	RI	MRGR	INT	2	1	Add Halfword
	AIP	75		MRGR	GRR	2	1	Add Indirect Pointer
	ARFA 0	161		RGEN	FIELD	-	-	Add R to FAR 0
	ARFA 1	171		RGEN	FIELD	-	-	Add R to FAR 1
	ARGT	000605		GEN	PCTLJ	6	5	Argument Transfer
	ATQ	135		AP	QUEUE	-	7	Add Entry to Top of Queue
	BCEQ	141602		BRAN	BRAN	-	-	Branch on Condition Code EQ
	BOGE	141605		BRAN	BRAN	-	-	Branch on Condition Code GE
	BOGT	141601		BRAN	BRAN	-	-	Branch on Condition Code GT
	BCLE	141600		BRAN	BRAN	-	-	Branch on Condition Code LE
	BCLT	141604		BRAN	BRAN	-	-	Branch on Condition Code LT
	BCNE	141603		BRAN	BRAN	-	-	Branch on Condition Code NE
	BCR	141705		BRAN	BRAN	-	-	Branch on CBIT Reset to 0
	BCS	141704		BRAN	BRAN	-	-	Branch on CBIT Set to 1
	BFEQ	122		IBRN	BRAN	-	4	Branch on F Equal to 0
	BFGE	125		IBRN	BRAN	-	4	Branch on F Greater Than or Equal to 0
	BFGT	121		IBRN	BRAN	-	4	Branch on F Greater Than 0
	BFLE	120		IBRN	BRAN	-	4	Branch on F Less Than or Equal to 0
	BFLT	124		IBRN	BRAN	-	4	Branch on F Less Than 0
	BFNE	123		IBRN	BRAN	-	4	Branch on F Not Equal to 0
	BHD1	144		IBRN	BRAN	-	-	Branch on r Decrementd by 1
	BHD2	145		IBRN	BRAN	-	-	Branch on r Decrementd by 2
	BHD4	146		IBRA	BRAN	-	-	Branch on r Decrementd by 4
	BHEQ	112		IBRN	BRAN	-	4	Branch on r Equal to 0
	BHGE	115		IBRN	BRAN	-	4	Branch on r Greater Than or Equal to 0
	BHGT	111		IBRN	BRAN	-	4	Branch on r Greater Than 0
	BHI1	140		IBRN	BRAN	-	-	Branch on r Incremented by 1
	BHI2	141		IBRN	BRAN	-	-	Branch on r Incremented by 2
	BHI4	142		IBRN	BRAN	-	-	Branch on r Incremented by 4
	BHLE	110		IBRN	BRAN	-	4	Branch on r Less Than or Equal to 0
	BHLT	114		IBRN	BRAN	-	4	Branch on r Less Than 0
	BHNE	113		IBRN	BRAN	-	4	Branch on r Not Equal to 0
	BLR	141707		BRAN	BRAN	-	-	Branch on LINK Reset to 0
	BLS	141706		BRAN	BRAN	-	-	Branch on LINK Set to 1
	BMEQ	141602		BRAN	BRAN	-	-	Branch on Magnitude Condition EQ
	BMGE	141706		BRAN	BRAN	-	-	Branch on Magnitude Condition GE

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	OC	Description
	BMGT	141710		BRAN	BRAN	-	-	Branch on Magnitude Condition GT
	BMLE	141711		BRAN	BRAN	-	-	Branch on Magnitude Condition LE
	BMLT	141707		BRAN	BRAN	-	-	Branch on Magnitude Condition LT
	BMNE	141603		BRAN	BRAN	-	-	Branch on Magnitude Condition NE
	BRBR	040-077		IBRN	BRAN	-	-	Branch on Register Bit Reset to 0
	BRBS	000-037		IBRN	BRAN	-	-	Branch on Register Bit Set to 1
	BRD1	134		IBRN	BRAN	-	-	Branch on R Decrement by 1
	BRD2	135		IBRN	BRAN	-	-	Branch on R Decrement by 1
	BRD4	136		IBRN	BRAN	-	-	Branch on R Decrement by 4
	BREQ	102		IBRN	BRAN	-	4	Branch on R Equal to 0
	BRGE	105		IBRN	BRAN	-	4	Branch on R Greater Than or Equal to 0
	BRGT	101		IBRN	BRAN	-	4	Branch on R Greater Than 0
	BRI1	130		IBRN	BRAN	-	-	Branch on R Incremented by 1
	BRI2	131		IBRN	BRAN	-	-	Branch on R Incremented by 2
	BRI4	132		IBRN	BRAN	-	-	Branch on R Incremented by 4
	BRLE	100		IBRN	BRAN	-	4	Branch on R Less Than or Equal to 0
	BRLT	104		IBRN	BRAN	-	4	Branch on R Less Than 0
	BRNE	103		IBRN	BRAN	-	4	Branch on R Not Equal to 0
	C	61	RI	MRGR	INT	1	1	Compare Fullword
	CALF	000705		AP	PCTLJ	6	5	Call Fault Handler
	CCP	45	R	GR	CPTR	-	1	Compare C Pointer
	CGT	026		RGEN	BRAN	6	5	Computed GOTO
	CH	71	RI	MRGR	INT	1	1	Compare Halfword
	CHS	040		RGEN	INT	-	-	Change Sign
	CMH	045		RGEN	LOGIC	-	-	Complement r
	CMR	044		RGEN	LOGIC	-	-	Complement R
	CR	056		RGEN	CLEAR	-	-	Clear R to 0
	CREL	062		RGEN	CLEAR	-	-	Clear R High Byte 1 Right
	CRER	063		RGEN	CLEAR	-	-	Clear R High Byte 2 Right
	CRHL	054		RGEN	CLEAR	-	-	Clear R Left Halfword
	CRHR	055		RGEN	CLEAR	-	-	Clear R Right Halfword
	CSR	041		RGEN	MOVE	5	-	Copy Sign of R
	D	62	RI	MRGR	INT	3	5	Divide Fullword
	DBLE	106		RGEN	FLPT	-	-	Convert Single to Double Precision Floating
	DCP	160		RGEN	CPTR	-	-	Decrement C Pointer
	DFA	15,17	RI	MRFR	FLPT	3	5	Double Precision Floating Add
	DFC	05,07	RI	MRFR	FLPT	-	1	Double Precision Floating Compare

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	DFCM	144		RGEN	FLPT	3	5	Double Precision Floating Complement
	DFD	31,33	RI	MRFR	FLPT	3	5	Double Precision Floating Divide
	DFL	01,03	RI	MRFR	FLPT	-	-	Double Precision Floating Load
	DFM	25,27	RI	MRFR	FLPT	3	5	Double Precision Floating Multiply
	DFS	21,23	RI	MRFR	FLPT	3	5	Double Precision Floating Subtract
	DFST	11,13		MRFR	FLPT	-	-	Double Precision Floating Store
	DH	72	RI	MRGR	INT	3	5	Divide Halfword
	DH1	130		RGEN	INT	2	1	Decrement r by 1
	DH2	131		RGEN	INT	2	1	Decrement r by 2
	DM	60		MRNR	INT	-	1	Decrement Memory Fullword
	DMH	70		MRNR	INT	-	1	Decrement Memory Halfword
	DR1	124		RGEN	INT	2	1	Decrement R by 1
	DR2	125		RGEN	INT	2	1	Decrement R by 2
	DRN	040300		GEN	FLPT	3	5	Double Round From Quad
	DRNM	140571		GEN	FLPT	8	5	Double Round From Quad Towards Negative Infinity
	DRNP	040301		GEN	FLPT	3	5	Double Round From Quad Towards Positive Infinity
	DRNZ	040302		GEN	FLPT	3	5	Double Round From Quad Towards Zero
	E16S	000011		GEN	ADMOD	-	-	Enter 16S Mode
	E32I	001010		GEN	ADMOD	-	-	Enter 32I Mode
	E32R	001013		GEN	ADMOD	-	-	Enter 32R Mode
	E32S	000013		GEN	ADMOD	-	-	Enter 32S Mode
	E64R	001011		GEN	ADMOD	-	-	Enter 64R Mode
	E64V	000010		GEN	ADMOD	-	-	Enter 64V Mode
	EAFA 0	001300		AP	FIELD	-	-	Effective Address to FAR 0
	EAFA 1	001310		AP	FIELD	-	-	Effective Address to FAR 1
	EALB	42		MRNR	PCTLJ	-	-	Effective Address to LB
	EAR	63		MRGR	PCTLJ	-	-	Effective Address to R
	EAXB	52		MRNR	PCTLJ	-	-	Effective Address to XB
R	EIO	34		MRGR	IO	-	7	Execute I/O
R	ENB	000401		GEN	IO	-	-	Enable Interrupts
R	ENBL	000401		GEN	IO	-	-	Enable Interrupts (Local)
R	ENEM	000400		GEN	IO	-	-	Enable Interrupts (Mutual)
R	ENBP	000402		GEN	IO	-	-	Enable Interrupts (Process)
	FA	014,16	RI	MRFR	FLPT	3	5	Floating Add
	FC	04,06	RI	MRFR	FLPT	-	1	Floating Compare
	FCDQ	140571		GEN	FLPT	-	-	Floating Convert Double to Quad
	FCM	100		RGEN	FLPT	3	5	Floating Complement

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	OC	Description
	FD	30,32	RI	MRFR	FLPT	3	5	Floating Divide
	FL	00,02	RI	MRFR	FLPT	-	-	Floating Load
	FLT	105,11		RGEN	FLPT	6	5	Convert Integer to Floating Point
	FLTH	102,11		RGEN	FLPT	6	5	Convert Halfword Integer to Floating Point
	FM	24,26	RI	MRFR	FLPT	3	5	Floating Multiply
	FRN	107		RGEN	FLPT	3	5	Floating Round
	FRNM	146		RGEN	FLPT	3	5	Floating Round Towards Negative Infinity
	FRNP	145		RGEN	FLPT	3	5	Floating Round Towards Positive Infinity
	FRNZ	147		RGEN	FLPT	3	5	Floating Round Towards Zero
	FS	20,22	RI	MRFR	FLPT	3	5	Floating Subtract
	FST	10,12		MRFR	FLPT	3	5	Floating Store
R	HLT	000000		GEN	MCTL	-	-	Halt
	I	41	R	MRGR	MOVE	-	-	Interchange R and Memory Fullword
	ICBL	065		RGEN	MOVE	-	-	Interchange Bytes and Clear Left
	ICBR	066		RGEN	MOVE	-	-	Interchange Bytes and Clear Right
	ICHL	060		RGEN	MOVE	-	-	Interchange Halfwords and Clear Left
	ICHR	061		RGEN	MOVE	-	-	Interchange Halfwords and Clear Right
	ICP	167		RGEN	CPTR	-	-	Increment C Pointer
	IH	51	R	MRGR	MOVE	-	-	Interchange r and and Memory Halfword
	IH1	126		RGEN	INT	2	1	Increment r by 1
	IH2	127		RGEN	INT	2	1	Increment r by 2
	IM	40		MRNR	INT	-	1	Increment Memory Fullword
	IMH	50		MRNR	INT	-	1	Increment Memory Halfword
R	INBC	001217		AP	PRCEX	6	5	Interrupt Notify Beginning, Clear Active Interrupt
R	INEN	001215		AP	PRCEX	6	5	Interrupt Notify Beginning
R	INEC	001216		AP	PRCEX	6	5	Interrupt Notify End, Clear Active Interrupt
R	INEN	001214		AP	PRCEX	6	5	Interrupt Notify End
R	INH	001001		GEN	IO	-	-	Inhibit Interrupts
R	INHL	001001		GEN	IO	-	-	Inhibit Interrupts (Local)
R	INHM	001000		GEN	IO	-	-	Inhibit Interrupts (Mutual)
R	INHP	001002		GEN	IO	-	-	Inhibit Interrupts (Process)
	INK	070		RGEN	KEYS	-	-	Input Keys
	INT	103,11		RGEN	FLPT	3	5	Convert Floating Point to Integer

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	INTH	101,11		RGEN	FLPT	3	5	Convert Floating Point to Halfword Integer
	IR1	122		RGEN	INT	2	1	Increment R by 1
	IR2	123		RGEN	INT	2	1	Increment R by 2
	IRB	064		RGEN	MOVE	-	-	Interchange r Bytes
	IRH	057		RGEN	MOVE	-	-	Interchange R Halves
R	IRTC	000603		GEN	IO	7	6	Interrupt Return, Clear Active Interrupt
R	IRTN	000601		GEN	IO	7	6	Interrupt Return
R	ITLB	000615		GEN	MCIL	6	5	Invalidate STLB Entry
	JMP	51		MRNR	PCILJ	-	-	Jump
	JSR	73		MRGR	PCILJ	-	-	Jump to Subroutine
	JSXB	61		MRNR	PCILJ	-	-	Jump and Save in XB
	L	01	RI	MRGR	MOVE	-	-	Load
	LCC	45		MRGR	CPIR	-	7	Load C Character
	LCEQ	153		RGEN	LTSTS	-	-	Load r on Condition Code EQ
	LOGE	154		RGEN	LTSTS	-	-	Load r on Condition Code GE
	LOGT	155		RGEN	LTSTS	-	-	Load r on Condition Code GT
	LGLE	151		RGEN	LTSTS	-	-	Load r on Condition Code LE
	LGLT	150		RGEN	LTSTS	-	-	Load r on Condition Code LT
	LCNE	152		RGEN	LTSTS	-	-	Load r on Condition Code NE
P	LDAR	44		MRGR	MOVE	-	5	Load from Addressed Register
	LDC 0	162		RGEN	CHAR	-	7	Load Character
	LDC 1	172		RGEN	CHAR	-	7	Load Character
	LEQ	003		RGEN	LTSTS	-	4	Load r on R Equal to 0
	LF	016		RGEN	LTSTS	-	5	Load False
	LFEQ	023,03		RGEN	LTSTS	-	4	Load r on F Equal to 0
	LFGE	024,03		RGEN	LTSTS	-	4	Load r on F Greater Than or Equal to 0
	LFGT	025,03		RGEN	LTSTS	-	4	Load r on F Greater Than 0
	LFLE	021,03		RGEN	LTSTS	-	4	Load r on F Less Than or Equal to 0
	LFLI 0	001303		BRAN	FIELD	-	-	Load FLR 0 Immediate
	LFLI 1	001313		BRAN	FIELD	-	-	Load FLR 1 Immediate
	LFLT	020,03		RGEN	LTSTS	-	4	Load r on F Less Than 0
	LFNE	022,03		RGEN	LTSTS	-	4	Load r on F Not Equal to 0
	LGE	004		RGEN	LTSTS	-	4	Load r on R Greater Than or Equal to 0
	LGT	005		RGEN	LTSTS	-	4	Load r on R Greater Than 0
	LH	11	RI	MRGR	MOVE	-	-	Load Halfword
	LHEQ	013		RGEN	LTSTS	-	4	Load r on r Equal to 0
	LHGE	004		RGEN	LTSTS	-	4	Load r on r Greater Than or Equal to 0
	LHGT	015		RGEN	LTSTS	-	4	Load r on r Greater Than 0
	LHL1	04	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 1

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	LHL2	14	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 2
	LHL3	35	R	MRGR	MOVE	-	-	Load Halfword Shifted Left by 3
	LHLE	011		RGEN	LTSTS	-	4	Load r on r Less Than or Equal to 0
	LHLT	000		RGEN	LTSTS	-	4	Load r on r Less Than 0
	LHNE	012		RGEN	LTSTS	-	4	Load r on r Not Equal to 0
R	LIOT	000044		AP	MCTL	6	5	Load IOTLB
	LIP	65		MRGR	GRR	-	-	Load Indirect Pointer
	LLE	001		RGEN	LTSTS	-	4	Load r on R Less Than or Equal to 0
	LLT	000		RGEN	LTSTS	-	4	Load r on R Less Than 0
	LNE	002		RGEN	LTSTS	-	4	Load r on R Not Equal to 0
R	LPID	000617		GEN	MCTL	-	-	Load Process ID
R	LPSW	000711		AP	MCTL	7	6	Load Process Status Word
	LT	017		RGEN	LTSTS	-	5	Load True
	M	42	RI	MRGR	INT	*	-	Multiply Fullword
	MH	52	RI	MRGR	INT	3	5	Multiply Halfword
	N	03	RI	MRGR	LOGIC	-	-	AND Fullword
R	NFYB	001211		AP	PRCEX	6	5	Notify
R	NFYE	001210		AP	PRCEX	6	5	Notify
	NH	13	RI	MRGR	LOGIC	-	-	AND Halfword
	NOP	000001		GEN	MCTL	-	-	No Operation
	O	23	RI	MRGR	LOGIC	-	-	OR Fullword
	OH	33	RI	MRGR	LOGIC	-	-	OR Halfword
	OTK	071		RGEN	KEYS	7	6	Output Keys
	PCL	41		MRNR	PCTLJ	6	5	Procedure Call
	PID	052		RGEN	INT	-	-	Position for Integer Divide
	PIDH	053		RGEN	INT	-	-	Position r for Integer Divide
	PIM	050		RGEN	INT	3	5	Position after Multiply
	PIMH	051		RGEN	INT	3	5	Position r after Multiply
	PRTN	000611		GEN	PCTLJ	7	6	Procedure Return
R	PTLB	000064		GEN	MCTL	6	5	Purge TLB
	QFAD	36		MRFR	FLPT	3	5	Quad Precision Floating Add
	QFC	47	RI	MRFR	FLPT	-	7	Quad Precision Floating Compare
	QFCM	140570		GEN	FLPT	3	5	Quad Precision Floating Complement
	QFDV	46		MRFR	FLPT	3	5	Quad Precision Floating Divide
	QFLD	34		MRFR	FLPT	-	-	Quad Precision Floating Load
	QFMP	45		MRFR	FLPT	3	5	Quad Precision Floating Multiply
	QFSB	37		MRFR	FLPT	3	5	Quad Precision Floating Subtract

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	QFST	35		MRFR	FLPT	-	-	Quad Precision Floating Store
	QINQ	140572		GEN	FLPT	3	5	Quad to Integer, in Quad
	QIQR	140573		GEN	FLPT	3	5	Quad to Integer, in Quad
	RBQ	133		AP	QUEUE	-	7	Convert Rounded Remove Entry from Bottom of Queue
R	RCB	140200		GEN	KEYS	8	-	Reset CBIT to 0
	RMC	000021		GEN	INIGY	-	-	Reset Machine Check Flag to 0
	ROT	24		MRGR	SHIFT	4	-	Rotate
	RRST	000717		AP	MCTL	-	-	Restore Registers
	RSAB	000715		AP	MCTL	-	-	Save Registers
	RTQ	132		RGEN	QUEUE	-	7	Remove Entry from Top of Queue
R	RTS	000511		GEN	MCTL	-	-	Reset Time Slice
	S	22	RI	MRGR	INT	2	1	Subtract Fullword
	SCB	140600		GEN	KEYS	5	-	Set CBIT to 1
	SOC	55		MRGR	CPTR	-	-	Store C Character
	SH	32	RI	MRGR	INT	2	1	Subtract Halfword
	SHA	15		MRGR	SHIFT	4	-	Shift Arithmetic
	SHL	05		MRGR	SHIFT	4	-	Shift Logical
	SHL1	076		RGEN	SHIFT	4	-	Shift R Left 1
	SHL2	077		RGEN	SHIFT	4	-	Shift R Left 2
	SHR1	120		RGEN	SHIFT	4	-	Shift R Right 1
	SHR2	121		RGEN	SHIFT	4	-	Shift R Right 2
	SL1	072		RGEN	SHIFT	4	-	Shift R Left 1
	SL2	073		RGEN	SHIFT	4	-	Shift R Left 2
	SR1	074		RGEN	SHIFT	4	-	Shift R Right 1
	SR2	075		RGEN	SHIFT	4	-	Shift R Right 2
	SSM	042		RGEN	INT	-	-	Set Sign Minus
	SSP	043		RGEN	INT	-	-	Set Sign Plus
	SSSN	040310		GEN	MCTL	6	5	Store System Serial Number
	ST	21		MRGR	MOVE	-	-	Store Fullword
P	STAR	54		MRGR	MOVE	-	5	Store into Addressed Register
	STC 0	166		RGEN	CHAR	-	7	Store Character
	STC 1	176		RGEN	CHAR	-	7	Store Character
	STCD	137		AP	MOVE	-	7	Store Conditional Fullword
	STCH	136		AP	MOVE	-	7	Store Conditional Halfword
	STEX	027		RGEN	PCTLJ	6	5	Stack Extend
	STFA 0	001320		AP	FIELD	-	-	Store FAR 0
	STFA 1	001330		AP	FIELD	-	-	Store FAR 1
	STH	31		MRGR	MOVE	-	-	Store Halfword
R	STPM	000024		GEN	MCTL	-	-	Store Processor Model Number
	STTM	000510		GEN	MCTL	6	5	Store Process Timer
	SVC	000505		GEN	PCTLJ	-	-	Supervisor Call
	TC	046		RGEN	INT	3	1	Two's Complement R
	TCH	047		RGEN	INT	3	1	Two's Complement r

Table D-2 (continued)
I Mode Instruction Summary

R	Mnem	Opcode	RI	Form	Func	C	CC	Description
	TCNP	76	R	MRNR	CPTR	-	1	Test C Null Pointer
	TFLR 0	163		RGEN	FIELD	-	-	Transfer FLR 0 to R
	TFLR 1	173		RGEN	FIELD	-	-	Transfer FLR 1 to R
	TM	44		MRNR	MCIL	-	1	Test Memory Fullword
	TMH	54		MRNR	INT	-	1	Test Memory Halfword
	TRFL 0	165		RGEN	FIELD	-	-	Transfer R to FLR 0
	TRFL 1	175		RGEN	FIELD	-	-	Transfer R to FLR 1
	TSTQ	104		RGEN	QUEUE	-	7	Test Queue
R	WAIT	000315		AP	PRCEX	-	-	Wait
	X	43	RI	MRGR	LOGIC	-	-	Exclusive OR Fullword
	XAD	001100		DECI	DECI	3	1	Decimal Add
	XBITD	001145		DECI	DECI	3	5	Binary to Decimal Conversion
	XCM	001102		DECI	DECI	-	1	Decimal Compare
	XDTB	001146		DECI	DECI	3	5	Decimal to Binary Conversion
	XDV	001107		DECI	DECI	3	5	Decimal Divide
	XED	001112		DECI	DECI	-	-	Numeric Edit
	XH	53	RI	MRGR	LOGIC	-	-	Exclusive OR Halfword
	XMP	001104		DECI	DECI	3	1	Decimal Multiply
	XMV	001101		DECI	DECI	3	1	Decimal Move
	ZCM	001117		CHAR	CHAR	6	7	Compare Character Field
	ZED	001111		CHAR	CHAR	-	-	Character Field Edit
	ZFIL	001116		CHAR	CHAR	6	5	Fill Field With Character
	ZM	43		MRNR	CLEAR	-	-	Clear Fullword
	ZMH	53		MRNR	CLEAR	-	-	Clear Halfword
	ZMV	001114		CHAR	CHAR	6	5	Move Character Field
	ZMVD	001115		CHAR	CHAR	6	5	Move Characters Between Equal Length Strings
	ZTRN	001110		CHAR	CHAR	-	-	Character String Translate

E

2455 Architecture

The 2455 processor has now been added to the Prime 50 Series computers. This new processor shares the architecture and operating system that is common to all 50 Series processors and makes the 50 Series a line of completely upward-compatible and downward-compatible systems.

The implementation of the common architecture, however, can be slightly different for each member of the 50 Series, allowing the different processors to address a wide variety of user needs while remaining compatible.

The architectural implementation of the 2455 is identical to that of the 2755 processor.

INDEX

A

Access rights,
 for segments, 4-16
 gate access, 8-7
 validation during memory
 access, 4-21
 values and their meanings,
 4-22

Address formation,
 DMx, 11-21

Address manipulation
 instructions, 6-9

Address translation,
 details of operation, 4-26
 mechanism, 4-26
 STLB, 1-4
 timing information, 4-24

Address traps,
 32R mode, 3-24
 64R mode, 3-27
 64V mode, 3-16
 action for 64V mode short,
 3-33, B-7
 discussion, 3-31
 register file correspondence,
 3-34

Addressing,
 address formation, 3-8
 components of virtual address,
 3-2
 direct, 3-8
 discussion, 3-1, B-6
 GRR, 3-10
 indexed, 3-8
 indirect, 3-8
 indirect indexed, 3-10
 instructions, 6-9
 modes, 3-10
 register file, 9-21
 traps (See Address traps)
 units of information, 3-1

Addressing modes,
 discussion, 3-10
 mnemonics table, 3-13
 summary table, 3-14

Air flow sensor, 10-18

Alignment,
 burst-mode DMA, 11-18
 burst-mode DMT, 11-20
 DMC control word, 11-19
 ECB in gate segments, 8-7
 PCB, 9-2, C-3
 QCB address, 11-21

Alignment (continued)

QCBs, 6-42
queues, 6-43

Architecture,
dual-stream, B-3
Prime 6350, 1-10
single-stream, 1-2

Argument pointers,
calculating and storing, 8-12
discussion, 8-6

Argument templates,
discussion, 8-6, 8-11
format, 8-6

Arithmetic logic unit, 1-5

Arithmetic overflow, 5-9

Arithmetic overflow instructions,
5-9

Arithmetic shift instructions,
6-14

Auxiliary base (XB),
alteration by PCL, 8-15
base register field, 3-7
indirect pointer calculation,
8-11
introduction, 3-4

B

Backward threaded stack frames,
8-3

Base registers,
discussion, 3-3, 3-7
format, 3-3
instructions, 6-9
relationship to offsets, 3-4

Battery backup capability, 1-10

Beat rate, 1-7

Beginning of list, 9-5

Binary numbers, 6-3, 6-4

Bit manipulation instructions,
6-2

Bits, 3-1

Boolean operations, 6-2

Branch cache, 1-9, 10-39

Branch instructions, 7-1

Breaks,
discussion, 10-1
summary of, 10-2

Burst-mode DMA, 11-18

Burst-mode DMT, 11-20

Bytes, 3-1

C

C language character pointer,
3-4, 3-11, 3-20

Cabinet overtemperature sensor,
10-18

Cache memory,
access details, 4-22
branch cache, 1-9
details of access, 4-19, B-10
discussion, 1-2, 1-10, 2-3,
4-14
entry format, 4-14, B-10
inhibiting use of, 4-17, 4-18
introduction, 1-2, B-2
invalidation by stream
synchronization unit, B-3
invalidation via IOTLB, 11-15,
B-28
sizes and hit rates, 2-3, B-5
use during address conversion,
4-2
virtual mapping, 4-22

Called procedure, 8-2

Callee, 8-2

Caller, 8-2

- Calling procedure, 8-2
- Calls, 8-1
- CBIT, 5-9
- Character manipulation,
 - examples, 6-39
 - field operation instructions, 6-17
 - instructions, 6-38
- Character strings,
 - as floating-point numbers, 6-26
 - instructions, 6-38
 - manipulation of, 6-39
- Checks,
 - diagnostic status words, 10-21
 - discussion, 10-18
 - handler, 10-18
 - handler operation, 10-35
 - header format, 10-21
 - MCM field, 10-34
 - reporting modes, 10-35
 - traps, 10-36
 - traps produced by checks and their actions, 10-37, B-26
 - types of, 10-18, 10-36
 - vectors, 10-21
- Checksum instructions, 6-2
- Clear register/memory instructions, 6-16
- Components of an instruction, 3-5
- Concealed stack, 10-10
- Concurrency control,
 - Prime 850 locks, B-3, B-5
- Condition codes, 5-9
- Control store, 1-4, B-2
- Control word format for decimal instructions, 6-34
- Controller,
 - address, 11-4
 - discussion, 11-1
 - relationship to processor, 11-1
- Controller address field, 11-4
- CPUNUM, C-3
- D
- Data movement instructions, 6-10
- Datatypes,
 - discussion, 6-1
 - summary with applicable I mode instructions, 6-49
 - summary with applicable R mode instructions, 6-47
 - summary with applicable S mode instructions, 6-47
 - summary with applicable V mode instructions, 6-47
- Decimal data,
 - accuracy, 6-36
 - control word format, 6-34
 - packed, 6-33
 - precision, 6-36
 - register use, 6-36
 - sign/digit representations for unpacked, 6-33
 - types, 6-35
 - unpacked, 6-32
- Descriptor Table Address
 - Register, 4-15, 4-29
- Diagnostic status words,
 - list of, 10-21
 - setting by multiple checks, 10-35
 - value after checks, 10-34
- Direct addressing, 3-8
- Direct memory access (See DMA)
- Direct memory access methods (See DMx)

Direct memory control, 11-19

Direct memory queue, 6-41, 6-45, 11-21

Direct memory transfer, 11-20

Dispatcher,
discussion, 9-16
operation, 9-27, B-18
operation on Prime 850, C-11

Displacement, 3-4, 3-7

DMA,
burst-mode, 11-18
discussion, 11-16
extended, 11-19
register file, 9-21
servicing a request, 11-17

DMC, 11-19

DMQ, 11-21
physical queues, 6-41
queue operations, 6-45

DMT, 11-20
burst-mode, 11-20

DMx,
address formation, 11-21
discussion, 11-10
DMA, 11-16
DMC, 11-19
DMQ, 11-21
DMT, 11-20
IOTLB, 11-14
mapped I/O, 11-13, B-27
transfer rates, 11-12

Double precision floating-point, 6-19

DSWPARITY,
format for Prime 2350 to 2755, 10-29
format for Prime 6350, 10-22
format for Prime 750, B-23
format for Prime 850, B-23
format for Prime 9650 and 9655, 10-29
format for Prime 9750 to 9955 II, 10-26

DSWPARITY2,
format for Prime 6350, 10-24

DSWPB, 10-34

DSWRMA, 10-33

DSWSTAT,
discussion, 10-36
format for earlier processors, B-25
format for Prime 2350 to 2755, 10-32
format for Prime 6350, 10-30
format for Prime 9650 and 9655, 10-32
format for Prime 9750 to 9955 II, 10-31

DTAR,
discussion, 4-15
format, 4-15
use during address translation, 4-29

Dual-stream architecture, B-3

E

Earlier processors,
address translation, 4-26, B-11
address trap action, 3-31, B-7
addressing, 3-1, B-6
altering sequential flow, 7-1, B-16
breaks, 10-1, B-19
cache, B-2
cache access, 4-19, B-10
cache entry format, B-10
cache sizes and hit rates, B-5
checks, 10-18, B-22
control store, B-2
datatypes, 6-1, B-11
dispatcher operation, 9-25, B-18
DMA register file, 9-21, B-18
DSWPARITY, B-23
DSWPB, 10-34, B-22
DSWRMA, 10-33, B-22
DSWSTAT, B-25
DTAR, 4-15, B-10

Earlier processors (continued)
 dual-stream architecture, B-3
 execution unit, 1-5, B-2
 faults, 10-6, B-22
 floating-point, 6-19, B-11
 HMAP, 4-18, B-10
 input/output, 11-1, B-27
 instruction stream units, B-3
 instruction unit, B-2
 interrupts, 10-3, B-19
 interval clock, B-27
 IOTLB, B-27
 keys, 5-4, B-11
 list of, 1-1, B-1
 memory management, 4-1, B-8
 microcode, B-2
 microcode register files, B-18
 modals, 5-2, B-11
 nonindexing 64V mode
 instructions, B-6
 physical and virtual memory,
 2-1, B-5
 procedure calls, 8-1, B-17
 process exchange, 9-1, B-17
 process exchange on Prime 850,
 C-1
 process interval timer, 9-25,
 B-18
 register files, B-17
 restricted instructions, 5-11,
 B-11
 SDT and SDW, 4-16, B-10
 single-stream architecture,
 1-2, B-2
 stacks, 8-1, B-17
 STLB, B-2
 STLB access, 4-19, B-10
 STLB entry format, B-8
 STLB hashing algorithm, B-9
 stream synchronization units,
 B-3
 system overview, 1-1, B-2
 traps, 10-37, B-26
 user register files, 9-19,
 B-18

ECB,

CALF instruction, 10-13
 discussion, 8-5
 format, 8-5
 gate segments, 8-7
 ring numbers, 8-7
 stack allocation, 8-10

ECL, 1-10

Effective address calculation
 instructions, 6-9

Embedded operating system, 8-1

Emitter coupled logic, 1-10

End of list, 9-5

Entry control block (See ECB)

Environment sensor support,
 check, 10-18
 discussion, 10-18

Environmental checks, 10-18

Excess 128, 6-19

Execution unit, B-2
 discussion, 1-5
 introduction, 1-5
 power-up initialization, A-1
 relationship to I/O controller,
 11-1

Exponent, 6-19

Extended DMA, 11-19

Extension segments, 8-2

F

FADDR, 10-16

FAR (See Field address register)

Fault address, 10-13

Fault bit, 4-16

Fault code, 10-13

Faults,
 access, 4-22, 8-7
 arithmetic exceptions, 10-16,
 B-22
 CALF instruction, 10-10
 classes, 10-6

Faults (continued)

- classes summary, 10-15
- concealed stack, 10-10
- decimal, 5-6
- discussion, 10-6
- floating-point, 5-6
- handler, 10-7
- integer, 5-6
- omitted argument pointer, 8-14
- page, 4-29
- PCB, 9-3
- pointer, 3-9, 8-11, 8-14
- process, 9-27, C-11
- SDW, 4-16
- semaphore overflow, 9-9, 9-13, 10-6, 10-9, 10-15
- servicing, 10-12
- stack overflow, 8-3, 8-10
- summary of, 10-6
- tables, 10-8
- vectors, 10-7

FOODE, 10-16, B-22

Field address register,

- format, 6-18
- instructions, 6-17
- introduction, 6-17
- overlap with floating-point registers, 6-17, 6-21, 9-20

Field length register,

- format, 6-18
- instructions, 6-17
- introduction, 6-17
- overlap with floating-point registers, 6-17, 6-21, 9-20

Field operations instructions, 6-17

Firmware, 1-4

Fixed-point data,

- addresses, 6-9
- discussion, 6-1
- field operations, 6-17
- instructions, 6-4, 6-10
- logical values, 6-2
- signed integers, 6-3

Flag bits in CALF stack frame, 10-13

Floating-point numbers,

- accumulators, 6-19
- accuracy, 6-26, B-14
- discussion, 6-19
- format, 6-20, B-12
- FORTTRAN 66 considerations, 6-26
- instructions, 6-22
- manipulation of, 6-23
- normalization, 6-23, 6-25, B-12
- precision, 6-26, B-15
- register overlap with field registers, 6-17, 6-21, 9-20
- rounding, 6-24, B-13
- zero, 6-23

FLR (See Field length register)

FORTTRAN 66 considerations, 6-26

Fraction, 6-19

Free pointer, 8-3

Function field, 11-4

G

Gate access, 8-7

Gate segments, 8-7

General register relative (See GRR)

General registers, alteration during procedure call, 8-15

GRR, 3-2, 3-4, 3-10, 3-11, 3-14, 3-21

GRR addressing, 3-10

Guard bits, 6-23, B-12

H

Halfwords, 3-1

- Hardware page map table, 4-18, 4-29
- Hashing algorithm (See STLB: hashing algorithm)
- Hit rate, 2-3
- HMAP,
 - discussion, 4-18
 - entry format, 4-18
 - use during address translation, 4-29
- Honeywell 316 and 516, 3-12
- I
- I mode,
 - behavior relating to 5-stage pipeline, 1-9
 - discussion, 3-11
 - performance, 1-9
- I/O,
 - discussion, 11-1
 - mapped, 11-13
- I/O Controller, 11-1, 11-4
- Immediate types, 3-20
- In Dispatcher bit, 9-29
- INA action, 11-9
- Index register,
 - discussion, 3-7
 - relationship to offsets, 3-4
- Indexed addressing, 3-8
- Indirect addressing,
 - argument templates, 8-6
 - calculation of pointers, 8-11
 - discussion, 3-8
 - format, 3-3, 3-4, 3-20
 - long form, 3-9
 - multiple levels, 3-8
 - pointers, 3-20, 8-6
 - relationship to offsets, 3-4
 - short form, 3-8
- Indirect bit,
 - 16S mode, 3-29
 - 32R mode, 3-24
 - 32S mode, 3-31
 - 64R mode, 3-27
 - discussion, 3-6
- Indirect indexed address, 3-10
- Indirection chain,
 - 32R mode, 3-24
 - 32S mode, 3-31
 - discussion, 3-8
 - involving indexing, 3-10
- Input/output,
 - discussion, 11-1
 - mapped, 11-13
- Instruction format,
 - 16S mode, 3-28
 - 32I mode, 3-20
 - 32R mode, 3-22
 - 32S mode, 3-30
 - 64R mode, 3-25
 - 64V mode long and indirect, 3-17
 - 64V mode short form, 3-15
 - typical, 3-6
- Instruction set,
 - address manipulation, 6-9
 - argument transfer, 8-14
 - arithmetic overflow, 5-9
 - bit manipulation, 6-2
 - branches, 7-1
 - character strings, 6-38
 - checksum, 6-2
 - clear register/memory, 6-16
 - conditional store, 6-13
 - conversion between fixed- and floating-point, 6-29
 - data movement, 6-10
 - datatypes, 6-1
 - deadlock prevention, 6-13
 - decimal, 6-37
 - decimal control word format, 6-34
 - effect address calculation, 6-9
 - EIO, 11-2
 - fast array reference, 6-9
 - fast decrement by one or two, 6-7

Instruction set (continued)

- fast increment by one or two, 6-4
- fast setting of bits in A, 6-7
- faults, 10-10
- fixed-point data, 6-10
- floating-point, 6-22
- floating-point accuracy, 6-27
- handling large integers, 6-4
- input/output, 11-2
- input/output operative actions, 11-9
- interrupt handling, 10-4
- interval clock, 10-47
- interval timer, 9-26
- invalidating IOTLB, 11-15
- jumps, 7-6
- keys, 5-8
- lock implementation, 6-13
- logic instructions, 6-2
- modals, 5-4
- overlapping strings, 6-39
- phantom interrupt, 10-4
- PIO, 11-2
- procedure call, 8-2
- process exchange, 9-7, 9-9
- process exchange on the Prime 850, C-6
- process timer, 9-26
- queues, 6-45, 6-46
- ready list, 9-13
- restricted instructions, 5-11
- results of comparisons, 5-9
- returning from procedures, 8-15
- semaphores, 9-7, 9-9
- semaphores on the Prime 850, C-6
- shift instructions, 6-14
- shifts versus rotates, 6-15
- signed integers, 6-4
- skips, 7-1
- special load/store, 6-13
- wait list, 9-9

Instruction stream,

- altering sequential flow, 7-1
- self-modifying code, 1-9
- storing data into, 1-9

Instruction stream units, B-3, C-1

Instruction unit, 1-2, 1-7, B-2

Integers, 6-3

Integrity,

- machine check, 5-4
- protection rings, 2-6

Interrupt response code, 10-3

Interrupts,

- disabling, 5-4
- discussion, 10-3
- enabling, 5-4
- external, 10-3
- inhibiting, 5-4
- memory increment, B-19
- response code, 11-11
- response time, 11-11
- standard, 5-4
- standard interrupt mode, B-21
- vectored, 5-4

Interval clock, 10-46, B-27

Interval timer, 9-25, B-18

Inward calls, 8-1, 8-7, 8-15

IOTLB,

- address format, 11-14
- discussion, 11-14, B-27
- entry format, 11-15, B-27
- mapping information, 11-14

IX mode (See GRR)

J

Jump instructions, 7-6

K

Keys,

- CALF stack frame, 10-13
- CBIT, 5-9
- condition codes, 5-9
- discussion, 5-4
- ECB, 8-5
- format in S and R modes, 5-5
- format in V and I modes, 5-6
- instructions, 5-8

Keys (continued)

LINK, 5-9
 PCL, 8-10
 PRTN, 8-15
 stack frame, 8-4
 undefined settings, 5-10

L

L bit, 8-6, 8-14

Last bit, 8-6, 8-14

LINK, 5-9

Link base (LB),

base register field, 3-7
 CALF stack frame, 10-13
 ECB, 8-5
 introduction, 3-4
 offset, 3-16
 PCL instruction, 8-10
 PRTN instruction, 8-15
 stack frame, 8-4

Load/store special instructions,
 6-13

Locks, B-3, B-5

Logic instructions, 6-2

Logical shift instruction, 6-14

Logical values, 6-2

Long form indirection, 3-9

M

Machine check,

discussion, 10-18
 recoverable (See Recoverable
 machine check)

Mapped I/O, 11-13, 11-17, B-27

Mask word for queues, 6-43

Master ISU, C-1

Memory,

cache, (See also Cache memory)
 data structures, 4-3
 details of access, 4-19, B-10
 details of address translation,
 4-26
 DTAR format, 4-15
 hardware page map table, 4-18
 interleaving, 2-4
 management, 4-1, B-8
 management data structures,
 4-3
 manager, 2-1
 page faults, 4-29
 parity error, 10-18
 physical (See Physical memory)
 segment descriptor word, 4-16
 timing information, 4-24, 4-26
 virtual (See Virtual memory)

Memory increment interrupt, B-19

Memory interleaving, 2-4

Memory manager, 2-1

Memory parity error, 10-18

Microcode, 1-4, B-2

Microcode register files,

for earlier processors, B-18
 for Prime 2350 to 2755, 9-24
 for Prime 6350, 9-22
 for Prime 9650 and 9655, 9-22
 for Prime 9750 to 9955 II,
 9-22

Microsecond timer, C-11

Missing memory module, 10-18

Modals,

discussion, 5-2
 format, 5-3
 instructions, 5-4
 MCM field, 10-34

N

Nonindexing instructions,

- 16S mode, 3-29
- 32R mode, 3-24
- 32S mode, 3-31
- 64R mode, 3-27
- 64V mode, 3-19, B-6

Normalization, 6-23, 6-25, B-12

Numbers, 6-3, 6-4

O

OCP action, 11-10

Offsets, 3-2, 3-4

Operating system,

- access via user programs, 2-5
- automatic shutdown, 10-18
- concealed stack, 10-12
- embedded, 8-1, 8-7, 8-15
- environment sensor support, 10-18
- gate segments, 8-7
- returning from inward calls, 8-15
- segmentation, 2-5
- UPS support, 10-18
- virtual memory management, 2-1

OTA action, 11-10

Overflow, 6-23

Overlap between field and floating-point registers, 6-17, 6-21, 9-20

OWNER, 9-20

OWNERH, 9-2, 9-20

P

Packed decimal data, 6-33

Page map table, 4-17, 4-29

Pages,

- discussion, 2-5
- disk vs. memory, 4-2
- hardware page map table, 4-18
- page fault vector, 10-8
- page faults, 4-29
- status checking during address translation, 4-29

PCB,

- concealed stack, 10-11
- discussion, 9-2
- fault vectors, 10-7
- format, 9-3
- format for Prime 850, C-4
- interval timer, 9-25, B-18
- OWNERH, 9-2, C-3
- Prime 850 dispatcher, C-12
- Prime 850 format, C-3
- PX lock, C-6
- wait list, 9-7

PCBA and PCBB, 9-5

Performance,

- burst-mode DMA, 11-18
- burst-mode DMT, 11-20
- character manipulation instructions, 6-38
- fast array reference instructions, 6-9
- fast decrement instructions, 6-7
- fast increment instructions, 6-4
- fast setting of bits in A, 6-7
- mapped I/O, 11-13, B-27
- pipeline flushing, 1-9
- public vs. private shared segments, 4-21
- Ring 0 memory access, 4-21

Phantom interrupt code, 10-4

Physical memory,

- addressing, 3-1
- conversion from virtual address, 4-2
- data structures, 4-3
- details of access, 4-19, B-10
- details of address translation, 4-26
- discussion, 2-2
- DTAR format, 4-15

- Physical memory (continued)
 - elements, 2-2
 - error detection and correction, 2-4, B-5
 - hardware page map table, 4-18
 - interleaving, 2-4, B-5
 - introduction, 2-1
 - packaging, 2-4, B-5
 - page faults, 4-29
 - pages, 2-3
 - segment descriptor word, 4-16
 - size of, 3-1
 - STLB, 2-8
 - timing information, 4-24, 4-26
 - translation from virtual memory, 4-1
- Physical queues, 6-41
- PIO,
 - communications controller
 - addresses, 11-7
 - controller address assignments, 11-5
 - controller ID numbers, 11-7
 - discussion, 11-2
 - EIO effect on condition codes, 11-10
 - instructions, 11-2
- Pipeline,
 - 2-phase, 1-9
 - 5-stage, 1-7
 - explicit flush by instruction
 - stream, 1-9
 - flushing, 1-9
 - handling invalidation via
 - branch cache, 1-9
 - introduction, 1-7
- PMT,
 - discussion, 4-17
 - entry format, 4-17
 - use during address translation, 4-29
- Pointer,
 - argument, 8-6
 - bit number, 3-9
 - discussion, 3-9
 - extension bit, 3-9
 - fault bit, 3-9
 - indirect, 8-6
- Postindexed addressing, 3-10
- Power-up,
 - initialization values, A-2
 - process, A-1
- PPA and PPB, 9-5
- Preindexed addressing, 3-10
- Prime 150 (See Earlier processors)
- Prime 2250 (See Earlier processors)
- Prime 2350 (See individual subjects)
- Prime 2450 (See individual subjects)
- Prime 250 (See Earlier processors)
- Prime 250-II (See Earlier processors)
- Prime 2550 (See individual subjects)
- Prime 2655 (See individual subjects)
- Prime 2755 (See individual subjects)
- Prime 350 (See Earlier processors)
- Prime 400 (See Earlier processors)
- Prime 450 (See Earlier processors)
- Prime 500 (See Earlier processors)
- Prime 550 (See Earlier processors)
- Prime 550-II (See Earlier processors)

Prime 6350 (See individual subjects)

Prime 650 (See Earlier processors)

Prime 750 (See Earlier processors)

Prime 850 (See Earlier processors)

Prime 9650 (See individual subjects)

Prime 9655 (See individual subjects)

Prime 9750 (See individual subjects)

Prime 9755 (See individual subjects)

Prime 9950 (See individual subjects)

Prime 9955 (See individual subjects)

Prime 9955 II (See individual subjects)

Prime I450 (See Earlier processors)

PRIMOS (See Operating system)

Priority headers, 9-5

Procedure base (PB),
base register field, 3-7
CALF stack frame, 10-13
introduction, 3-3
PCL instruction, 8-10

Procedure control block (See PCB)

Procedures,
address of current link frame,
3-4
address of current stack frame,
3-4

Procedures (continued)

address of currently active
procedure, 3-3
affected registers, 8-15
argument transfer instruction,
8-14
details of calling, 8-7
discussion, 8-1
ECB, 8-5
gate segments, 8-7
inward calls, 8-7
PCL instruction, 8-2
returning to caller, 8-15
stack management, 8-2
types of calls, 8-1

Process exchange mechanism,
affecting break handling, 10-2
affecting interrupt handling,
10-4
check handler operation, 10-35
discussion, 9-1, C-1
dispatcher, 9-16, 9-27, B-18
dispatcher operation, C-11
dual-stream processors, C-1
example of ready list use, 9-6
fault servicing, 10-12
instructions, 9-9
interval timer, 9-25, B-18
NOTIFY on Prime 850, C-9
OWNER, 9-20
OWNERH, 9-2, 9-20
PCB, 9-2
PCBA and PCBB, 9-5
PPA and PPB, 9-5
priority headers, 9-5
PX lock, C-3
ready list, 9-2
register files, 9-17
semaphores, 9-7
wait list, 9-7

Processes,
dispatcher, 9-16, 9-27, B-18
fault vectors, 10-7
implementation on single-stream
processors, 9-1
instructions for scheduling,
9-9
interval timer, 9-25, B-18
introduction, 8-1
PCB, 9-2
process exchange mechanism,
9-1

Processes (continued)
 process exchange on Prime 850,
 C-1
 register files (See User
 register files)
 semaphores, 9-7
 wait list, 9-7

Processor board overtemperature
 sensor, 10-18

Program counter,
 relationship to PB, 3-4
 transferring control, 8-1

Programmed I/O (See PIO)

Protection rings, 2-6, 3-2

Pure procedure, 1-9

PX lock, C-3, C-6

PXM (See Process exchange
 mechanism)

Q

QCB,
 alignment, 6-42
 discussion, 6-41
 format, 6-42

Quad precision floating-point,
 6-19

Queue control block, 6-41, 6-42

Queues,
 algorithms, 6-44
 discussion, 6-41
 instructions, 6-45, 6-46
 mask word, 6-43
 maximum number of elements,
 6-44
 physical, 6-41
 Prime 850 locks, B-5
 virtual, 6-41

R

R mode,
 behavior relating to 5-stage
 pipeline, 1-9
 discussion, 3-12
 index limitations, 3-8
 input/output, 11-2
 introduction, 3-12
 performance, 1-9

Ready list,
 data base, 9-5
 discussion, 9-2
 example, 9-6
 example with associated PCB
 lists, 9-4
 instructions, 9-13
 Prime 850, C-6

Recoverable machine check,
 10-18, 10-36, 10-37

Register file,
 actions during interrupt
 handling, 10-3
 allocation for 2350 to 2755,
 9-17
 allocation for 6350, 9-16
 allocation for 9650 and 9655,
 9-17
 allocation for 9750 to 9955 II,
 9-16
 allocation for earlier
 processors, B-17
 arithmetic exceptions, 10-16,
 B-22
 check handling by processor,
 10-34
 decimal instructions, 6-36
 direct addressing, 9-21
 DMA channels, 9-21, 11-16
 floating-point registers, 6-19
 interval timer in dispatcher,
 9-27, B-18
 manipulation by dispatcher,
 9-28
 microcode scratch for earlier
 processors, B-18
 microcode scratch for Prime
 2350 to 2755, 9-24
 microcode scratch for Prime
 6350, 9-22

Register file (continued)
 microcode scratch for Prime
 9650 and 9655, 9-24
 microcode scratch for Prime
 9750 to 9955 II, 9-22
 NOTIFY instruction, 9-13
 Prime 850, C-10
 Prime 850 dispatcher, C-12
 register-to-register
 instructions, 6-13
 relationship to processor, 1-5
 restoring, 6-13
 save by NOTIFY instruction,
 9-13
 saving, 6-13
 short save by WAIT instruction,
 9-9
 TIMERH and TIMERL, C-11
 use by dispatcher, 9-27, B-18
 user processes (See User
 register files)
 WAIT instruction, 9-9

Register overlap between field
 and floating-point registers,
 6-17, 6-21, 9-20

Restricted instructions,
 discussion, 5-1
 list of, 5-11

Result of the chain, 3-8

Ring 0,
 queues, 6-42
 restricted instructions, 5-1

Ring 2, 4-21

Ring numbers,
 calculation during procedure
 call, 8-7
 calculation during procedure
 return, 8-15
 discussion, 3-2
 queues, 6-42
 restricted instructions, 5-1
 undefined results, 4-21
 weakening during memory access,
 4-21

Rings of protection, 2-6, 3-2

Rotate instructions, 6-14

Rounding, 6-24, B-13

S

S bit, 8-6, 8-11, 8-15

S mode,
 behavior relating to 5-stage
 pipeline, 1-9
 discussion, 3-12
 index limitations, 3-8
 input/output, 11-2
 introduction, 3-12
 performance, 1-9

Save Done bit, 9-28, C-12

SDT, 4-16, 4-29

SDW, 4-16

Sector,
 addressing current, 3-29, 3-31
 discussion, 3-12

Security and protection rings,
 2-6

Segment descriptor table,
 discussion, 4-16
 use during address translation,
 4-29

Segment descriptor word,
 discussion, 4-16
 entry format, 4-16

Segment numbers,
 discussion, 3-2
 use during address translation,
 4-29

Segment Table Lookaside Buffer
 (See STLB)

Segment Table Origin Register,
 4-15, 4-29

Segmentation and STLB, 1-4

- Segments,
 - access rights, 4-16
 - CALF stack frame stack root, 10-13
 - dedicated to PCB, 9-2
 - descriptor words, 4-16
 - discussion, 2-5
 - faults, 4-29
 - gate access, 8-7
 - numbers, 3-2
 - protection rings, 2-6
 - segment fault handling, 10-8
 - segmented mode, 3-16
 - shared, 2-5, 4-19
 - stack extension, 8-2
 - stack root, 8-4, 8-5
 - transferring program control
 - between, 7-1, 7-6
 - unshared, 2-5
 - use of segment 0 for check vectors, 10-21
 - use of segment 4 for check headers, 10-21
- Self-modifying code, 1-9
- Semaphores, 9-7
- Shared subsystem implementation via segmentation, 2-5
- Shift instructions, 6-14, 6-15
- Short form indirection, 3-8
- Signed integers,
 - formats, 6-3
 - instructions, 6-4
- Single precision floating-point, 6-19
- Single-stream architecture, 1-2, B-2
- Skip instructions, 7-1
- SKS action, 11-10
- Slave ISU, C-1
- Stack,
 - allocation, 8-10
 - allocation of argument pointers, 8-14
 - argument transfer instruction, 8-14
 - caller's state saved, 8-10
 - concealed, 10-10
 - deallocation by returning, 8-15
 - discussion, 8-2
 - ECB, 8-5
 - extension pointer, 8-3
 - extension segments, 8-2
 - frame format, 8-4
 - frame size, 8-5
 - frames, 8-3
 - header, 8-2
 - stack root, 8-2, 8-4, 8-5
- Stack base (SB),
 - base register field, 3-7
 - CALF stack frame, 10-13
 - introduction, 3-4
 - stack allocation, 8-10
 - stack deallocation, 8-15
 - stack frame, 8-4
- Standard interrupt mode, B-21
- STLB,
 - access details, 4-19, 4-24
 - details of access, 4-19, B-10
 - discussion, 1-4, 1-10, 2-8, B-2
 - entry format, 4-5, B-8
 - hashing algorithm for earlier processors, B-9
 - hashing algorithm for Prime 2350 to 2655, 4-12
 - hashing algorithm for Prime 2755, 4-10
 - hashing algorithm for Prime 6350, 4-7
 - hashing algorithm for Prime 9650 and 9655, 4-12
 - hashing algorithm for Prime 9750 to 9950, 4-9
 - hashing algorithm for Prime 9955 and 9955 II, 4-7
 - IOTLB, 11-14, B-27
 - use during address conversion, 4-2
 - use during procedure call, 8-7

Store bit, 8-6, 8-11, 8-15

Stream synchronization unit, B-3

String manipulation,
examples, 6-38
field operation instructions,
6-17
instructions, 6-38

Subroutines (See Procedures)

Syndrome bits,
discussion, 10-42
for Prime 6350, 10-42
for Prime 9750 to 9955 II,
10-43
for rest of 50 series, 10-44

System overview, 1-1, B-2

T

Tag modifier, 3-20

Time-sharing (See Process
exchange mechanism)

Traps,
access violation, 10-41
cache parity error, 10-40
discussion, 10-37
DMx, 10-44
error correcting code, 10-42
fetch cycle, 9-30, 10-45,
10-46
hard parity error, 10-40
machine check, 10-44
missing memory module, 10-41
page modified, 10-41
read address, 10-41
restricted instruction, 10-45
software breaks, 10-45
STLB miss, 10-41
STLB parity error, 10-40
types and priorities, 10-37,
B-26
write address, 10-44

U

Underflow, 6-23

Unpacked decimal data,
discussion, 6-32
format, 6-32
sign/digit representations for,
6-33

UPS support, 10-18

User register files,
discussion, 9-17
mnemonics used, 9-18
overlap between field and
floating-point registers,
6-17, 6-21, 9-20
OWNER, 9-20
structure, 9-19

V

V mode,
behavior relating to 5-stage
pipeline, 1-9
discussion, 3-11
index limitations, 3-8
input/output, 11-2
performance, 1-9

Virtual address format, 3-3, 4-2

Virtual memory,
addressing, 3-1, 4-1
conversion to physical address,
4-2
data structures, 4-3
details of access, 4-19, B-10
details of address translation,
4-26
discussion, 2-5
DTAR format, 4-15
hardware page map table, 4-18
introduction, 2-1
page faults, 4-29
pages, 2-5
protection rings, 2-6
segment descriptor word, 4-16
segments, 2-5
size of, 2-5, 3-1
space, 2-5

Virtual memory (continued)
 STLB, 2-8
 timing information, 4-24, 4-26
 translation to physical memory,
 4-1
 use of disks, 2-5

Virtual pages, 2-5

Virtual queues, 6-41

W

Wait list, 9-7 to 9-9, C-6

Words, 3-1

X

X register, 3-7, 8-11

Y

Y register, 3-7

Z

Zero, 6-23

SURVEY

READER RESPONSE FORM

DOC9473-2LA System Architecture Reference Guide Second Edition

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

Readability: ___hard to understand ___average ___very clear

Technical level: ___too simple ___about right ___too technical

Technical accuracy: ___poor ___average ___very good

Examples: ___too many ___about right ___too few

Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

4. What faults or errors gave you problems? _____

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:



Attention: Technical Publications
Bldg 10
Prime Park, Natick, Ma. 01760



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES





00C9473-2LA